THE RELATIVE PERFORMANCE OF SCALABLE LOAD BALANCING ALGORITHMS IN LOOSELY-COUPLED DISTRIBUTED SYSTEMS

VOL II

Rupert Anthony Simpson

Submitted for the degree of Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

July 1994

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

APPENDIX A

LOAD BALANCING PROTOCOLS: MESSAGE TRACING

A.1 ROUTING TABLES

This section presents the message routing tables generated using Node0 as the initial starting point for traversing the network.

Processor0 PRIMARY route

dest.:0	via::node-1	distance::16 hops
dest.:1	via::node1	distance::1 hops
dest.:2	via::node1	distance::2 hops
dest.:3	via::node1	distance::3 hops
dest.:4	via::node4	distance::1 hops
dest.:5	via::node1	distance::2 hops
dest.:6	via::node4	distance::3 hops
dest.:7	via::node4	distance::4 hops
dest.:8	via::node4	distance::2 hops
dest.:9	via::node1	distance::3 hops
dest.:10	via::node1	distance::4 hops
dest.:11	via::node4	distance::5 hops
dest.:12	via::node4	distance::3 hops
dest.:13	via::node1	distance::4 hops
dest.:14	via::node1	distance::5 hops
dest.:15	via::node4	distance::6 hops

5 ROUTES via this node

Processor1 PRIMARY route

dest.:0	via::node0	distance::1 hops
dest.:1	via::node-1	distance::16 hops
dest.:2	via::node2	distance::1 hops
dest.:3	via::node2	distance::2 hops
dest.:4	via::node0	distance::2 hops
dest.:5	via::node5	distance::1 hops
dest.:6	via::node2	distance::2 hops
dest.:7	via::node2	distance::3 hops
dest.:8	via::node0	distance::3 hops
dest.:9	via::node5	distance::2 hops
dest.:10	via::node5	distance::3 hops
dest.:11	via::node2	distance::4 hops
dest.:12	via::node0	distance::4 hops
dest.:13	via::node5	distance::3 hops
dest.:14	via::node2	distance::4 hops
dest.:15	via::node2	distance::5 hops

12 ROUTES via this node ***

Table A.1 Message Routing Tables for Node0 and Node1

Processor2 PRIMARY route

dest.:0	via::node1	distance::2 hops
dest.:1	via::node1	distance::1 hops
dest.:2	via::node-1	distance::16 hops
dest.:3	via::node3	distance::1 hops
dest.:4	via::node1	distance::3 hops
dest.:5	via::node6	distance::2 hops
dest.:6	via::node6	distance::1 hops
dest.:7	via::node3	distance::2 hops
dest.:8	via::node1	distance::4 hops
dest.:9	via::node1	distance::3 hops
dest.:10	via::node6	distance::2 hops
dest.:11	via::node6	distance::3 hops
dest.:12	via::node6	distance::5 hops
dest.:13	via::node6	distance::4 hops
dest.:14	via::node6	distance::3 hops
dest.:15	via::node3	distance::4 hops

11 ROUTES via this node

Processor3 PRIMARY route

dest.:0	via::node2	distance::3 hops
dest.:1	via::node2	distance::2 hops
dest.:2	via::node2	distance::1 hops
dest.:3	via::node-1	distance::16 hops
dest.:4	via::node7	distance::4 hops
dest.:5	via::node7	distance::3 hops
dest.:6	via::node7	distance::2 hops
dest.:7	via::node7	distance::1 hops
dest.:8	via::node2	distance::5 hops
dest.:9	via::node2	distance::4 hops
dest.:10	via::node7	distance::3 hops
dest.:11	via::node7	distance::2 hops
dest.:12	via::node7	distance::6 hops
dest.:13	via::node7	distance::5 hops
dest.:14	via::node7	distance::4 hops
dest.:15	via::node7	distance::3 hops

6 ROUTES via this node

Table A.2 Message Routing Tables for Node2 and Node3

Processor4	PRIMARY	route
dest.:0	via::node0	distance::1 hops
dest.:1	via::node5	distance::2 hops
dest.:2	via::node0	distance::3 hops
dest.:3	via::node5	distance::4 hops
dest.:4	via::node-1	distance::16 hops
dest.:5	via::node5	distance::1 hops
dest.:6	via::node5	distance::2 hops
dest.:7	via::node5	distance::3 hops
dest.:8	via::node8	distance::1 hops
dest.:9	via::node8	distance::2 hops
dest.:10	via::node8	distance::3 hops
dest.:11	via::node8	distance::4 hops
dest.:12	via::node8	distance::2 hops
dest.:13	via::node8	distance::3 hops
dest.:14	via::node5	distance::4 hops
dest.:15	via::node5	distance::5 hops

9 ROUTES via this node ***

Processor5	PRIMARY	route
dest.:0	via::node4	distance::2 hops
dest.:1	via::node1	distance::1 hops
dest.:2	via::node6	distance::2 hops
dest.:3	via::node6	distance::3 hops
dest.:4	via::node4	distance::1 hops
dest.:5	via::node-1	distance::16 hops
dest.:6	via::node6	distance::1 hops
dest.:7	via::node6	distance::2 hops
dest.:8	via::node9	distance::2 hops
dest.:9	via::node9	distance::1 hops
dest.:10	via::node9	distance::2 hops
dest.:11	via::node9	distance::3 hops
dest.:12	via::node9	distance::3 hops
dest.:13	via::node9	distance::2 hops
dest.:14	via::node9	distance::3 hops
dest.:15	via::node9	distance::4 hops

14 ROUTES via this node

Table A.3 Message Routing Tables for Node4 and Node5

Processor6	PRIMARY	route
dest.:0	via::node5	distance::3 hops
dest.:1	via::node5	distance::2 hops
dest.:2	via::node2	distance::1 hops
dest.:3	via::node7	distance::2 hops
dest.:4	via::node5	distance::2 hops
dest.:5	via::node5	distance::1 hops
dest.:6	via::node-1	distance::16 hops
dest.:7	via::node7	distance::1 hops
dest.:8	via::node10	distance::3 hops
dest.:9	via::node10	distance::2 hops
dest.:10	via::node10	distance::1 hops
dest.:11	via::node10	distance::2 hops
dest.:12	via::node10	distance::4 hops
dest.:13	via::node10	distance::3 hops
dest.:14	via::node10	distance::2 hops
dest.:15	via::node10	distance::3 hops

13 ROUTES via this node ***

Processor7	PRIMARY	route
dest.:0	via::node3	distance::4 hops
dest.:1	via::node3	distance::3 hops
dest.:2	via::node3	distance::2 hops
dest.:3	via::node3	distance::1 hops
dest.:4	via::node6	distance::3 hops
dest.:5	via::node6	distance::2 hops
dest.:6	via::node6	distance::1 hops
dest.:7	via::node-1	distance::16 hops
dest.:8	via::node11	distance::4 hops
dest.:9	via::node11	distance::3 hops
dest.:10	via::node11	distance::2 hops
dest.:11	via::node11	distance::1 hops
dest.:12	via::node11	distance::5 hops
dest.:13	via::node11	distance::4 hops
dest.:14	via::node11	distance::3 hops
dest.:15	via::node11	distance::2 hops

12 ROUTES via this node ***

Table A.4 Message Routing Tables for Node6 and Node7

Processor8 PRIMARY route

dest.:0	via::node4	distance::2 hops
dest.:1	via::node9	distance::3 hops
dest.:2	via::node9	distance::4 hops
dest.:3	via::node9	distance::5 hops
dest.:4	via::node4	distance::1 hops
dest.:5	via::node9	distance::2 hops
dest.:6	via::node9	distance::3 hops
dest.:7	via::node9	distance::4 hops
dest.:8	via::node-1	distance::16 hops
dest.:9	via::node9	distance::1 hops
dest.:10	via::node9	distance::2 hops
dest.:11	via::node9	distance::3 hops
dest.:12	via::node12	distance::1 hops
dest.:13	via::node12	distance::2 hops
dest.:14	via::node12	distance::3 hops
dest.:15	via::node12	distance::4 hops

12 ROUTES via this node ***

Processor9 PRIMARY route

dest.:0	via::node8	distance::3 hops
dest.:1	via::node5	distance::2 hops
dest.:2	via::node10	distance::3 hops
dest.:3	via::node10	distance::4 hops
dest.:4	via::node8	distance::2 hops
dest.:5	via::node5	distance::1 hops
dest.:6	via::node10	distance::2 hops
dest.:7	via::node10	distance::3 hops
dest.:8	via::node8	distance::1 hops
dest.:9	via::node-1	distance::16 hops
dest.:10	via::node10	distance::1 hops
dest.:11	via::node10	distance::2 hops
dest.:12	via::node13	distance::2 hops
dest.:13	via::node13	distance::1 hops
dest.:14	via::node13	distance::2 hops
dest.:15	via::node13	distance::3 hops

23 ROUTES via this node ***

Table A.5 Message Routing Tables for Node8 and Node9

Processor10 PRIMARY route

dest.:0	via::node9	distance::4 hops
dest.:1	via::node9	distance::3 hops
dest.:2	via::node6	distance::2 hops
dest.:3	via::node11	distance::3 hops
dest.:4	via::node9	distance::3 hops
dest.:5	via::node9	distance::2 hops
dest.:6	via::node6	distance::1 hops
dest.:7	via::node11	distance::2 hops
dest.:8	via::node9	distance::2 hops
dest.:9	via::node9	distance::1 hops
dest.:10	via::node-1	distance::16 hops
dest.:11	via::node11	distance::1 hops
dest.:12	via::node14	distance::3 hops
dest.:13	via::node14	distance::2 hops
dest.:14	via::node14	distance::1 hops
dest.:15	via::node14	distance::2 hops

23 ROUTES via this node ***

Processor11 PRIMARY route

dest.:0	via::node10	distance::5 hops
dest.:1	via::node10	distance::4 hops
dest.:2	via::node10	distance::3 hops
dest.:3	via::node7	distance::2 hops
dest.:4	via::node10	distance::4 hops
dest.:5	via::node10	distance::3 hops
dest.:6	via::node10	distance::2 hops
dest.:7	via::node7	distance::1 hops
dest.:8	via::node10	distance::3 hops
dest.:9	via::node10	distance::2 hops
dest.:10	via::node10	distance::1 hops
dest.:11	via::node-1	distance::16 hops
dest.:12	via::node15	distance::4 hops
dest.:13	via::node15	distance::3 hops
dest.:14	via::node15	distance::2 hops
dest.:15	via::node15	distance::1 hops

16 ROUTES via this node ***

Table A.6 Message Routing Tables for Node10 and Node11

Processor12 PRIMARY route

dest.:0	via::node8	distance::3 hops
dest.:1	via::node13	distance::4 hops
dest.:2	via::node13	distance::5 hops
dest.:3	via::node8	distance::6 hops
dest.:4	via::node8	distance::2 hops
dest.:5	via::node13	distance::3 hops
dest.:6	via::node13	distance::4 hops
dest.:7	via::node8	distance::5 hops
dest.:8	via::node8	distance::1 hops
dest.:9	via::node13	distance::2 hops
dest.:10	via::node13	distance::3 hops
dest.:11	via::node13	distance::4 hops
dest.:12	via::node-1	distance::16 hops
dest.:13	via::node13	distance::1 hops
dest.:14	via::node13	distance::2 hops
dest.:15	via::node13	distance::3 hops

7 ROUTES via this node ***

Processor13 PRIMARY route

dest.:0	via::node12	distance::4 hops
dest.:1	via::node9	distance::3 hops
dest.:2	via::node14	distance::4 hops
dest.:3	via::node14	distance::5 hops
dest.:4	via::node12	distance::3 hops
dest.:5	via::node9	distance::2 hops
dest.:6	via::node14	distance::3 hops
dest.:7	via::node14	distance::4 hops
dest.:8	via::node12	distance::2 hops
dest.:9	via::node9	distance::1 hops
dest.:10	via::node14	distance::2 hops
dest.:11	via::node14	distance::3 hops
dest.:12	via::node12	distance::1 hops
dest.:13	via::node-1	distance::16 hops
dest.:14	via::node14	distance::1 hops
dest.:15	via::node14	distance::2 hops

20 ROUTES via this node ***

Table A.7 Message Routing Tables for Node12 and Node13

Processor14 PRIMARY route

dest.:0	via::node13	distance::5 hops
dest.:1	via::node13	distance::4 hops
dest.:2	via::node10	distance::3 hops
dest.:3	via::node15	distance::4 hops
dest.:4	via::node13	distance::4 hops
dest.:5	via::node13	distance::3 hops
dest.:6	via::node10	distance::2 hops
dest.:7	via::node15	distance::3 hops
dest.:8	via::node13	distance::3 hops
dest.:9	via::node13	distance::2 hops
dest.:10	via::node10	distance::1 hops
dest.:11	via::node15	distance::2 hops
dest.:12	via::node13	distance::2 hops
dest.:13	via::node13	distance::1 hops
dest.:14	via::node-1	distance::16 hops
dest.:15	via::node15	distance::1 hops

18 ROUTES via this node ***

Processor15 PRIMARY route

dest.:0	via::node14	distance::6 hops
dest.:1	via::node11	distance::5 hops
dest.:2	via::node11	distance::4 hops
dest.:3	via::node11	distance::3 hops
dest.:4	via::node14	distance::5 hops
dest.:5	via::node14	distance::4 hops
dest.:6	via::node11	distance::3 hops
dest.:7	via::node11	distance::2 hops
dest.:8	via::node14	distance::4 hops
dest.:9	via::node14	distance::3 hops
dest.:10	via::node11	distance::2 hops
dest.:11	via::node11	distance::1 hops
dest.:12	via::node14	distance::3 hops
dest.:13	via::node14	distance::2 hops
dest.:14	via::node14	distance::1 hops
dest.:15	via::node-1	distance::16 hops

7 ROUTES via this node ***

Table A.8 Message Routing Tables for Node14 and Node15

A.2 ALGORITHMS

A.2.1 The Threshold Policy

The simulation model was run for the threshold policy with debugging information enabled. Statistics were obtained for the whole system and for each host every 500 seconds of simulated time. The following message dialogue was recorded shortly after the first phase of the simulation:

500.001.29152.90263.86202.2819 1.44 0.3155 7.3187 1.83

Node10 receives Packet N3-N10:: Node3 -overloaded- has asked node10 TO accept
Node10 GMT 500.040 TripT0.256 TIMER:WAITP_TOUT Etime: 500.296

Node10 receives Packet N3-N10:: node 10 RECEIVES process6396 [load = 2] FROM node 3

Node8 GMT 500.277 TripT0.260 TIMER:PRB_TOUT Etime: 500.537
Node8 receives Packet N3-N8:: Node8::Time:500.582725 Node3's reply (prc6398) was 3
Node8 GMT 500.584 TripT0.254 TIMER:PRB_TOUT Etime: 500.838

Node10 receives Packet N8-N10:: Node8 -overloaded- has asked node10 TO accept
Node10 GMT 500.612 TripT0.254 TIMER:WAITP_TOUT Etime: 500.866

Node8 receives Packet N10-N8::
Node8::Time:500.735934 Node10's reply (prc6398) was -2
node 8 sends process6398 [load = 6] to node 10

Node10 receives Packet N8-N10:: node 10 RECEIVES process6398 [load = 3] FROM node 8
Node10 receives Packet N6-N10:: Node6 -overloaded- has asked node10 TO accept

Node3 probes node10 which subsequently accepts prc6396 from the latter. Node8 probes Node3 and node10 on behalf of prc6398 where the former rejects whilst the latter is willing to accept. A further request made by Node6 is rejected as Node10 is no longer underloaded.

Node8 receives Packet N9-N8:: node 8 receives Exit msg [prc:: 6392 at node:9]
Node8 receives Packet N11-N8:: node 8 receives Exit msg [prc:: 6393 at node:11]
Node8 GMT 502.410 TripT0.256 TIMER:PRB_TOUT Etime: 502.666
Node8 receives Packet N10-N8:: node 8 receives Exit msg [prc:: 6398 at node:10]
Node8 receives Packet N11-N8:: Node8::Time:502.585341 Node11's reply (prc6422) was 3

Node8 receives exit messages for processes 6392, 6393, and 6398 from nodes *9, 11,* and *10* respectively. In addition, Node8 also probes Node11 and receives a rejection reply in return.

A.2.2 Communicating Set Threshold Policy

The simulation model was run using the same job stream as the Threshold policy but using the communicating set variation. The output produced by the model after the first period was:

```
SIMULATION no.1 STARTED ON Tue May 24 08:53:29 1994

*** erand48 seeds are: 17757, 22851, and 10394 ***

No. of Processors =      16  Load value        = 0.80
Total No. of jobs =   350000Ld Balancing Alg. = Threshold L/B Policy

Threshold Level  =    2.00  Probe Limit     =      3
Convergence to < 2.00%


  TIME        VAR.  LOAD         DIFF.  RTime  %Conv        MIG.    Tx   %JOB

  ----        ----   ----   -----   -----   -----   ----   --   ----
 500.00     1.4233 2.9512 4.0840 2.3243   2.43  0.3094 7.7293 1.83
```

N: 0	Mg:152.00	L: 3	Rt: 2.28	Tx: 16	Imm: 121	Dth: 431
N: 1	Mg:171.00	L: 3	Rt: 2.45	Tx: 19	Imm: 145	Dth: 433
N: 2	Mg:146.00	L: 3	Rt: 2.35	Tx: 17	Imm: 160	Dth: 393
N: 3	Mg:148.00	L: 5	Rt: 2.50	Tx: 14	Imm: 156	Dth: 401
N: 4	Mg:159.00	L: 3	Rt: 2.35	Tx: 20	Imm: 163	Dth: 403
N: 5	Mg:156.00	L: 3	Rt: 2.17	Tx: 23	Imm: 178	Dth: 388
N: 6	Mg:156.00	L: 3	Rt: 2.44	Tx: 25	Imm: 155	Dth: 376
N: 7	Mg:167.00	L: 2	Rt: 2.73	Tx: 25	Imm: 141	Dth: 400
N: 8	Mg:120.00	L: 4	Rt: 2.18	Tx: 23	Imm: 163	Dth: 361
N: 9	Mg:152.00	L: 3	Rt: 2.12	Tx: 36	Imm: 165	Dth: 389
N: 10	Mg:173.00	L: 3	Rt: 2.26	Tx: 35	Imm: 176	Dth: 412
N: 11	Mg:150.00	L: 2	Rt: 2.30	Tx: 27	Imm: 169	Dth: 377
N: 12	Mg:172.00	L: 3	Rt: 2.40	Tx: 17	Imm: 152	Dth: 416
N: 13	Mg:142.00	L: 2	Rt: 2.59	Tx: 34	Imm: 147	Dth: 370
N: 14	Mg:153.00	L: 3	Rt: 2.41	Tx: 32	Imm: 159	Dth: 388
N: 15	Mg:158.00	L: 4	Rt: 2.59	Tx: 24	Imm: 125	Dth: 410

Figure A.1 shows the average response time for each node after the first period of the simulation run. Nodes 5, 8, and 9 attained good response times, whilst node 7 produced the worse response time.

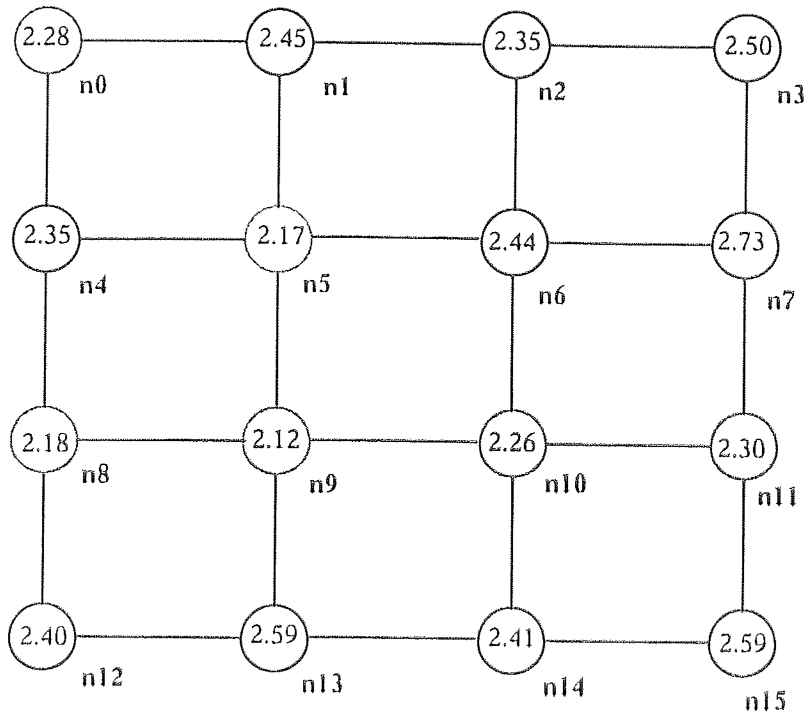


Figure A.1 Response Time After 500 Seconds using Cset Policy

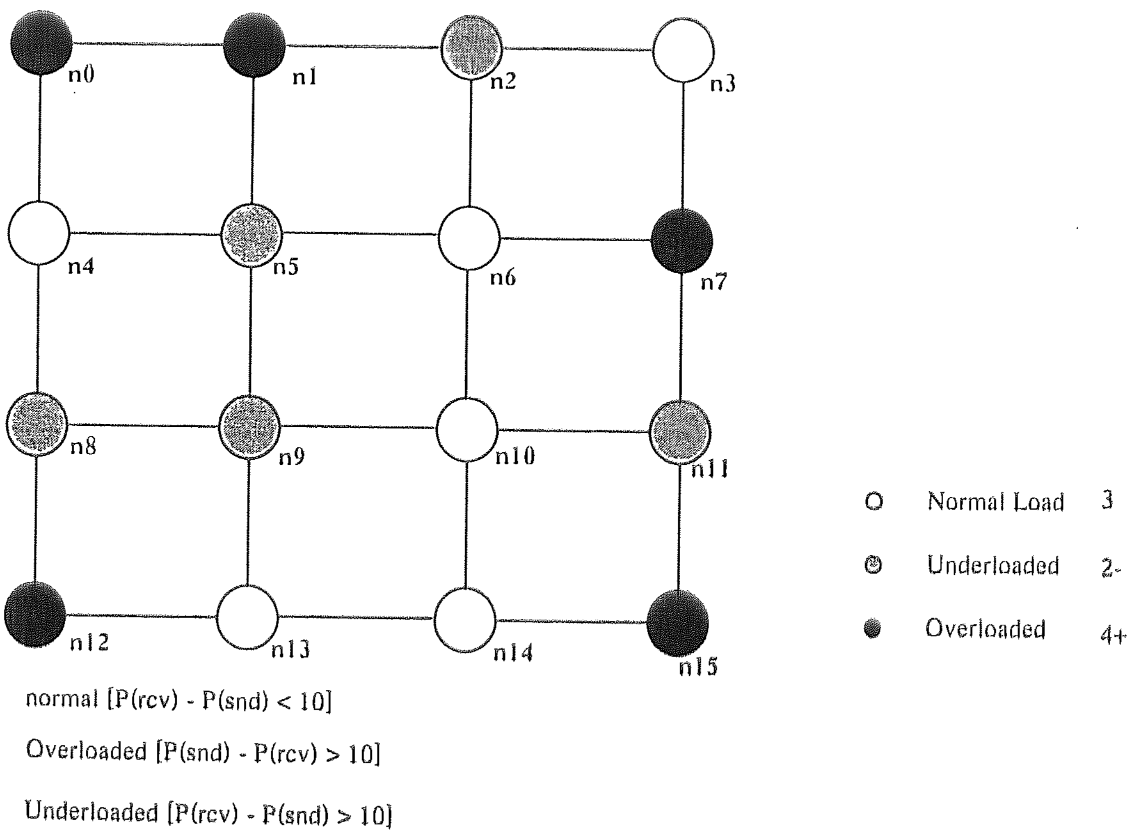


Figure A.2 Workload Distribution After 500 Seconds using Cset Policy

In Figure A.2 the load distribution in the network based on the ratio of processes sent and received is displayed.

The communicating set is built up as a by-product of load balancing events such as the receipt of or reply to a probe message. In this regard, node10's view of the network in its overloaded state after the first period of the simulation run only identifies nodes *9* as an underloaded site for fruitful load sharing activities. In contrast, communicating set of Node8 iis empty as it is unable to find an underloaded site.

N: 10 Mg:173.00 L: 3 Rt: 2.26 Tx: 35 Imm: 176 Dth: 412
CommSet:[9->2]

N: 8 Mg:120.00 L: 4 Rt: 2.18 Tx: 23 Imm: 163 Dth: 361
CommSet:[]

For ease of explanation and comprehension only the message dialogue of node*10* and node*8* is recorded. The dialogue is picked up three seconds before the end of the first simulation phase:

```
Node10 receives Packet N7-N10::   node 10 receives Exit msg [prc:: 6318 at node:7]
Node10 receives Packet N0-N10::   node 10 RECEIVES process6347 [load = 3] FROM node 0
Node10 receives Packet N0-N10::   Node0 -overloaded- has asked node10 TO accept
Node10 GMT 497.333       TripT0.258     TIMER:WAITP_TOUT       Etime: 497.591
Node10 receives Packet N7-N10::   node 10 receives Exit msg [prc:: 6320 at node:7]
Node10 receives Packet N0-N10::   node 10 RECEIVES process6354 [load = 3] FROM node 0
Node10 receives Packet N14-N10::  Node14 -overloaded- has asked node10 TO accept
```

An exit message is received by node10 from node7 for one of its original processes (prc6318) followed by process 6347 from node0. As node10 is a member of the communicating set for node0 a further probe message is sent by the latter and the subsequent migration of prc6354 to node10. A further probe message by node14 was rejected as node10 is no longer underloaded.

```
Node8 receives Packet N6-N8::     node 8 receives Exit msg [prc:: 6341 at node:6]
Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept
Node8 GMT 497.570        TripT0.252     TIMER:WAITP_TOUT       Etime: 497.822
Node8 receives Packet N12-N8::    node 8 RECEIVES process6356 [load = 2] FROM node 12
Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept
Node8 GMT 497.644        TripT0.252     TIMER:WAITP_TOUT       Etime: 497.896
Node8 receives Packet N12-N8::    node 8 RECEIVES process6359 [load = 3] FROM node 12
Node10 GMT 498.111       TripT0.254     TIMER:PRB_TOUT   Etime: 498.365
Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept
```

Node8 receives consecutive probe messages from node12 resulting in the migration of processes 6356 and 6359 from the latter to the former. As node8 is a member of its communicating set a further probe message is sent. However, this will be rejected as Node8 is no longer underloaded.

Node10 receives Packet N7-N10:: Node10::Time:498.313249 Node7's reply (prc6364) was 3
Node10 GMT 498.314 TripT0.252 TIMER:PRB_TOUT Etime: 498.566
Node10 receives Packet N14-N10:: Node10::Time:498.365658 Node14's reply (prc6364) was 4
Node10 GMT 498.367 TripT0.252 TIMER:PRB_TOUT Etime: 498.619
Node10 receives Packet N14-N10:: Node14 -overloaded- has asked node10 TO accept

Node10 receives Packet N9-N10::
Node10::Time:498.424092 Node9's reply (prc6364) was -2
node 10 sends process6364 [load = 4] to node 9

As Node10 is now overloaded, its communicating set will be empty. That is, it would only contain information on overloaded sites that have sent it probe messages. Therefore, remote sites 7 and 14 are probed at random and both were found to be overloaded. A further probe message sent to node9 was successful resulting in the migration of prc6364.

Node8 receives Packet N9-N8:: Node9 -overloaded- has asked node8 TO accept
Node8 receives Packet N14-N8:: node 8 receives Exit msg [prc:: 6344 at node:14]
Node8 receives Packet N9-N8:: Node9 -overloaded- has asked node8 TO accept
Node8 GMT 499.383 TripT0.256 TIMER:PRB_TOUT Etime: 499.639
Node8 GMT 499.487 TripT0.256 TIMER:PRB_TOUT Etime: 499.743
Node8 receives Packet N14-N8:: Node8::Time:499.588688 Node14's reply (prc6386) was 4
Node8 GMT 499.590 TripT0.252 TIMER:PRB_TOUT Etime: 499.842
Node8 GMT 499.593 TripT0.254 TIMER:PRB_TOUT Etime: 499.847

Node8 receives Packet N4-N8::
Node8::Time:499.645715 Node4's reply (prc6386) was -2
node 8 sends process6386 [load = 5] to node 4

In its previous overloaded state node8 performed a successful migration (prc6344) to node14. Therefore, given its current state two probe messages are sent to node14 on behalf of the excess local processes. However, these requests are rejected and another two consecutive probe messages were sent to randomly selected hosts. Node4 had available capacity and process 6386 is migrated to it.

Node10 receives Packet N14-N10:: Node14 -overloaded- has asked node10 TO accept

Node10 receives Packet N0-N10:: Node0 -overloaded- has asked node10 TO accept
Node10 GMT 499.207 TripT0.258 TIMER:WAITP_TOUT Etime: 499.465
Node10 receives Packet N0-N10:: Node0 -overloaded- has asked node10 TO accept
Node10 GMT 499.456 TripT0.258 TIMER:WAITP_TOUT Etime: 499.714
Node10 receives Packet N0-N10:: node 10 RECEIVES process6384 [load = 3] FROM node 0

Node10 receives Packet N0-N10:: node 10 RECEIVES process6388 [load = 3] FROM node 0

Probe messages are sent by node14 and node0 to node10. However, at the time of node14's enquiry, the recipient was overloaded. Therefore, in the case of node0, the two requests made were accepted and process 6384 and 6388 migrated.

Node8 GMT 499.659 TripT0.252 TIMER:PRB_TOUT Etime: 499.911
Node8 receives Packet N14-N8:: Node8::Time:499.710342 Node14's reply (prc6391) was 4
Node8 GMT 499.711 TripT0.252 TIMER:PRB_TOUT Etime: 499.963

Node8 receives Packet N0-N8:: Node8::Time:499.712351 Node0's reply (prc6392) was 3

Node8 GMT 499.713 TripT0.252 TIMER:PRB_TOUT Etime: 499.965
Node8 receives Packet N4-N8:: Node8::Time:499.766364 Node4's reply (prc6393) was 3
Node8 GMT 499.767 TripT0.256 TIMER:PRB_TOUT Etime: 500.023
Node8 receives Packet N4-N8:: Node8::Time:499.818773 Node4's reply (prc6391) was 3
Node8 GMT 499.820 TripT0.256 TIMER:PRB_TOUT Etime: 500.076
Node8 receives Packet N4-N8:: Node8::Time:499.820782 Node4's reply (prc6392) was 3
Node8 GMT 499.822 TripT0.258 TIMER:PRB_TOUT Etime: 500.080

Node8 receives Packet N14-N8:: Node8::Time:499.923591 Node14's reply (prc6393) was 3
Node8 GMT 499.925 TripT0.258 TIMER:PRB_TOUT Etime: 500.183

500.00 1.42332.95124.08402.3243 2.43 0.3094 7.7293 1.83

Node8 receives Packet N15-N8:: Node8::Time:500.026400 Node15's reply (prc6392) was 4
Node8 receives Packet N1-N8:: Node8::Time:500.027405 Node1's reply (prc6391) was 3
Node8 receives Packet N2-N8:: Node8::Time:500.179610 Node2's reply (prc6393) was 3
Node8 GMT 500.281 TripT0.254 TIMER:PRB_TOUT Etime: 500.535

In this case node8 sends consecutive requests to the only member of its communicating set (node4) on behalf of processes 6391 to 6393, but receives in turn, consecutive rejections. This is also the case for requests sent to nodes *14, 0, 15, 1,* and 2.

Node10 receives Packet N9-N10:: node 10 receives Exit msg [prc:: 6364 at node:9]
Node8 receives Packet N5-N8:: Node8::Time:500.484019 Node5's reply (prc6398) was 3
Node8 GMT 500.485 TripT0.256 TIMER:PRB_TOUT Etime: 500.741
Node10 receives Packet N6-N10:: Node6 -overloaded- has asked node10 TO accept

Node8 receives Packet N11-N8::
Node8::Time:500.637228 Node11's reply (prc6398) was -2
node 8 sends process6398 [load = 5] to node 11

Node8 receives Packet N4-N8:: node 8 receives Exit msg [prc:: 6386 at node:4]
Node8 receives Packet N15-N8:: Node15 -overloaded- has asked node8 TO accept

Both nodes 8 and 10 receive probe messages which they reject in turn as they are not in an underloaded state. However, node8 on its second attempt, found node11 willing to accept prc6398.

Node10 receives Packet N0-N10:: Node0 -overloaded- has asked node10 TO accept
Node10 GMT 501.421 TripT0.258 TIMER:WAITP_TOUT Etime: 501.679
Node10 receives Packet N0-N10:: node 10 RECEIVES pr6412 [load = 2] FROM node 0

Node8 receives Packet N6-N8:: Node6 -overloaded- has asked node8 TO accept
Node8 receives Packet N3-N8:: Node3 -overloaded- has asked node8 TO accept
Node8 receives Packet N11-N8:: node 8 receives Exit msg [prc:: 6398 at node:11]

Node10 receives Packet N4-N10:: Node4 -overloaded- has asked node10 TO accept
Node10 GMT 502.188 TripT0.256 TIMER:WAITP_TOUT Etime: 502.444

Node10 receives Packet N4-N10:: node 10 RECEIVES prc6415 [load = 3] FROM node 4

Node8 GMT 502.383 TripT0.256 TIMER:PRB_TOUT Etime: 502.639
Node8 receives Packet N11-N8::
Node8::Time:502.585832 Node11's reply (prc6420) was -2
node 8 sends process6420 [load = 4] to node 11

As node10 is in an underloaded state the probes made to it by node0 and node4 were successful resulting in process migrations. Likewise, Node8 in an overloaded state also successfully probes the last node (nodell) to receive a process from it.

Node10 receives Packet N4-N10:: Node4 -overloaded- has asked node10 TO accept
Node10 GMT 503.261 TripT0.256 TIMER:WAITP_TOUT Etime: 503.517
Node10 receives Packet N4-N10:: node 10 RECEIVES prc6432 [load = 3] FROM node 4

Node10 receives Packet N4-N10:: Node4 -overloaded- has asked node10 TO accept

Node8 receives Packet N11-N8:: node 8 receives Exit msg [prc:: 6420 at node:11]
Node8 receives Packet N4-N8:: Node4 -overloaded- has asked node8 TO accept
Node8 GMT 504.033 TripT0.252 TIMER:WAITP_TOUT Etime: 504.285
Node8 receives Packet N4-N8:: node 8 RECEIVES prc6437 [load = 3] FROM node 4

Node10 receives a further process from node4 but rejects its next request. Finally, node8 in in the underloaded state and accepts process 6437 from node4.

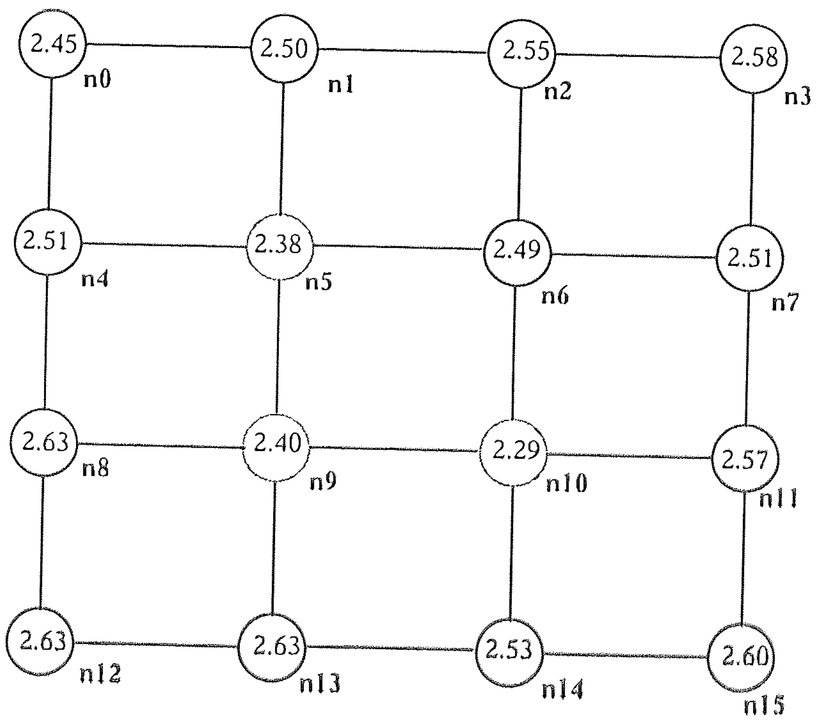
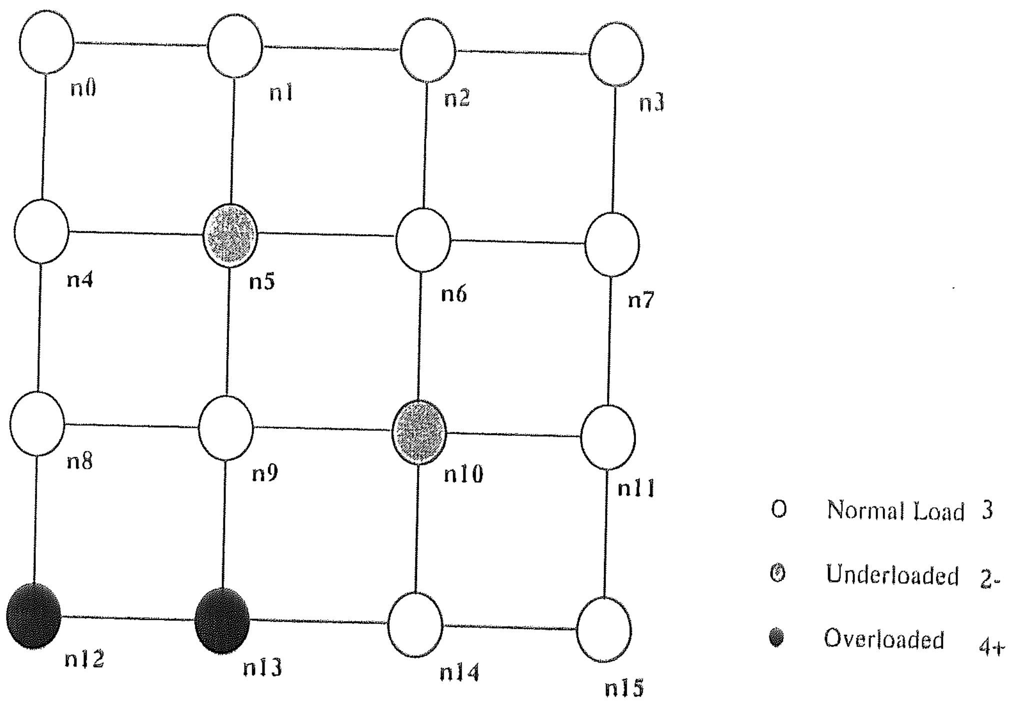


Figure A.3 Response Times After 4000 Seconds using Cset Policy

Figure A.3 shows the runtime performance after 4000 seconds of simulation time. Whilst the average response time has increased overall, the run-time performance of nodes 3, 7, and 13 have improved. However, nodes *5, 9,* and *10* continue to yield better than average performance despite the increase in response times.

In terms of their load state after 4000 seconds of simulated time, the following output was produced:

N: 0	Mg:1156.00	L: 2	Rt: 2.45	Tx:: 15	Imm:1004	Dth: 3274
N: 1	Mg:1184.00	L: 3	Rt: 2.50	Tx:: 19	Imm:1207	Dth: 3257
N: 2	Mg:1200.00	L: 2	Rt: 2.55	Tx:: 18	Imm:1243	Dth: 3200
N: 3	Mg:1154.00	L: 5	Rt: 2.58	Tx:: 15	Imm:1178	Dth: 3179
N: 4	Mg:1197.00	L: 4	Rt: 2.51	Tx:: 21	Imm:1242	Dth: 3213
N: 5	Mg:1270.00	L: 3	Rt: 2.38	Tx:: 27	Imm:1329	Dth: 3272
N: 6	Mg:1217.00	L: 1	Rt: 2.49	Tx:: 28	Imm:1229	Dth: 3208
N: 7	Mg:1255.00	L: 3	Rt: 2.51	Tx:: 24	Imm:1254	Dth: 3197
N: 8	Mg:1211.00	L: 3	Rt: 2.63	Tx:: 25	Imm:1186	Dth: 3184
N: 9	Mg:1286.00	L: 3	Rt: 2.40	Tx:: 38	Imm:1309	Dth: 3229
N: 10	Mg:1235.00	L: 2	Rt: 2.29	Tx:: 38	Imm:1359	Dth: 3135
N: 11	Mg:1185.00	L: 1	Rt: 2.57	Tx:: 31	Imm:1212	Dth: 3092
N: 12	Mg:1186.00	L: 4	Rt: 2.63	Tx:: 19	Imm:1135	Dth: 3236
N: 13	Mg:1240.00	L: 2	Rt: 2.63	Tx:: 38	Imm:1160	Dth: 3191
N: 14	Mg:1232.00	L: 2	Rt: 2.53	Tx:: 36	Imm:1193	Dth: 3226
N: 15	Mg:1213.00	L: 2	Rt: 2.60	Tx:: 21	Imm:1181	Dth: 3215



Overloaded [P(snd) - P(rcv) >> 50]

Underloaded [P(rcv) - P(snd) >>50]

Figure A.4 Workload Distribution After 4000 Seconds using Cset Policy

On the basis of the above table, nodes 3, *4, and 12* are overloaded and nodes *0, 2, 6, 10, 11, 13, 14, 15* underloaded. However, the ratio of the processes received and transmitted identifies nodes *5* and *10* to be underloaded and nodes *12, and 13* to be overloaded. This is illustrated in Figure A.4.

A.2.3 Modified Communicating Set Threshold Policy

Given the weakness of thr previous communicating set implementation in handling clusters of process arrivals, the algorithm was modified such that random probes are selected until a reply is received for an initial probe to a member of the communicating set.

The output produced by the model after the first period was:

```
SIMULATION no.1 STARTED ON Tue May 24 08:53:29 1994

*** erand48 seeds are: 17757, 22851, and 10394 ***

No. of Processors =      16  Load value      = 0.80
Total No. of jobs =  350000Ld Balancing Alg. = Threshold L/B Policy

Threshold Level  =    2.00  Probe Limit     =     3
Convergence to < 2.00%

TIME        VAR.  LOAD          DIFF.  RTime  %Conv        MIG.    Tx   %JOB
----        ----  ----   -----  -----  -----  ----    --   ----
500.00     1.3745 2.9074 3.9800 2.2420   3.00  0.3241 7.4719 1.83

   N:  0  Mg:170.00   L:  5  Rt: 2.38    Tx:  14    Imm: 126    Dth:  431
   N:  1  Mg:177.00   L:  3  Rt: 2.19    Tx:  17    Imm: 160    Dth:  433
   N:  2  Mg:156.00   L:  3  Rt: 2.28    Tx:  17    Imm: 149    Dth:  392
   N:  3  Mg:169.00   L:  6  Rt: 2.25    Tx:  11    Imm: 175    Dth:  400
   N:  4  Mg:159.00   L:  2  Rt: 2.30    Tx:  20    Imm: 158    Dth:  403
   N:  5  Mg:164.00   L:  3  Rt: 2.19    Tx:  23    Imm: 181    Dth:  387
   N:  6  Mg:159.00   L:  3  Rt: 2.25    Tx:  23    Imm: 173    Dth:  377
   N:  7  Mg:177.00   L:  4  Rt: 2.29    Tx:  20    Imm: 173    Dth:  398
   N:  8  Mg:143.00   L:  6  Rt: 2.39    Tx:  19    Imm: 188    Dth:  361
   N:  9  Mg:165.00   L:  3  Rt: 2.05    Tx:  32    Imm: 178    Dth:  388
   N: 10  Mg:166.00   L:  1  Rt: 2.30    Tx:  38    Imm: 158    Dth:  412
   N: 11  Mg:160.00   L:  3  Rt: 2.21    Tx:  24    Imm: 179    Dth:  378
   N: 12  Mg:166.00   L:  4  Rt: 2.44    Tx:  16    Imm: 154    Dth:  415
   N: 13  Mg:145.00   L:  3  Rt: 2.58    Tx:  33    Imm: 151    Dth:  370
   N: 14  Mg:156.00   L:  4  Rt: 2.38    Tx:  30    Imm: 166    Dth:  387
   N: 15  Mg:161.00   L:  4  Rt: 2.50    Tx:  23    Imm: 124    Dth:  408
```

Figure A.5 shows the average response time for each node after the first period of the simulation run. Nodes 1, 5, and 9 attained good response times, whilst nodes 12, 13, and 15 produced the worse response times.

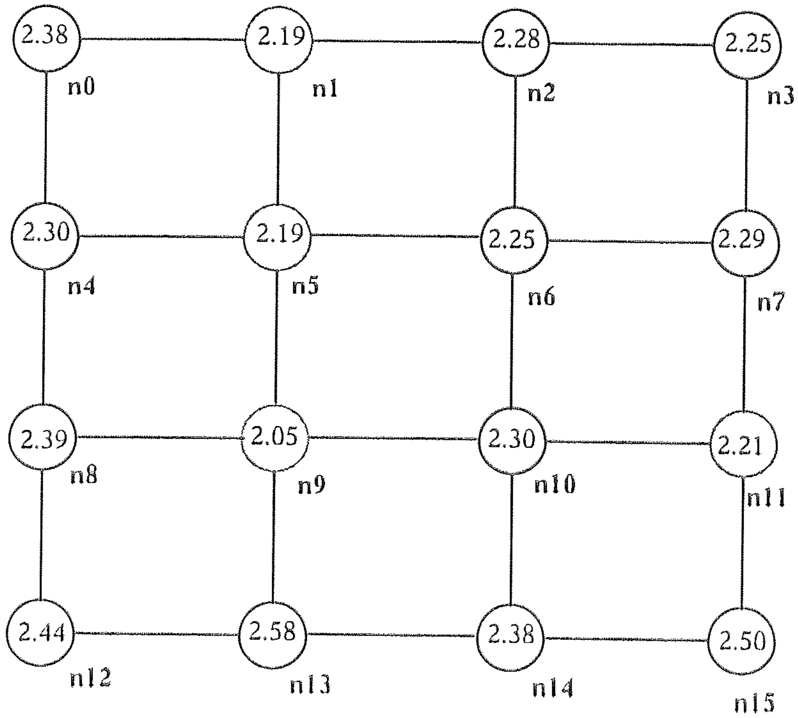
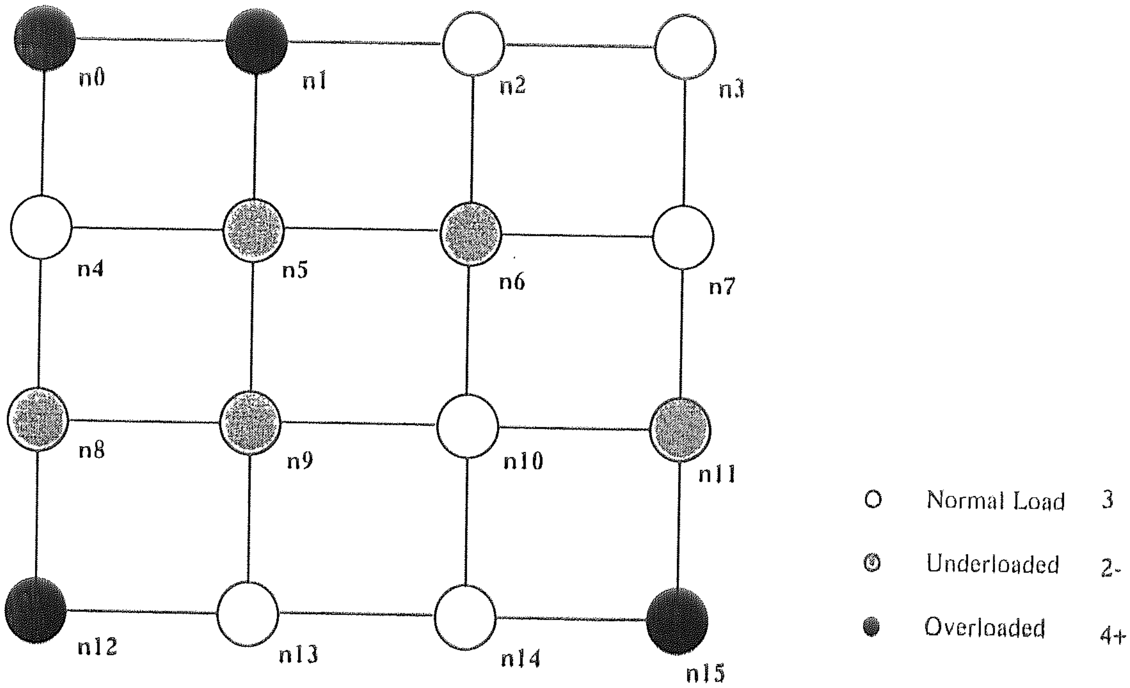


Figure A.5 Response Times After 500 Seconds using Modified Cset Implementation



normal [P(rcv) - P(snd) < 10]

Overloaded [P(snd) - P(rcv) > 10]

Underloaded [P(rcv) - P(snd) > 10]

Figure A.6 Workload After 500 Seconds (Modified Cset Implementation)

Figure A.6 shows the load distribution in the network based on the ratio of processes sent and received.

The following trace focussed on the dialogue for node*8* in handling load balancing in cases where processes arrive in clusters.

```
Node8 receives Packet N9-N8::     Node9 -overloaded- has asked node8 TO accept
Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept

Node8 GMT 499.386         TripT0.252    TIMER:PRB_TOUT    Etime: 499.638
Node8 receives Packet N9-N8::     Node8::Time:499.489505    Node9's reply (prc6387) was 3

Node8 GMT 499.491         TripT0.254    TIMER:PRB_TOUT    Etime: 499.745
Node8 GMT 499.492         TripT0.256    TIMER:PRB_TOUT    Etime: 499.748
Node8 GMT 499.597         TripT0.256    TIMER:PRB_TOUT    Etime: 499.853

Node8 receives Packet N13-N8::    Node8::Time:499.648742    Node13's reply (prc6387) was 4
```

Four process arrivals at node8 resulted in the the random selection of four remote hosts to which probe messages were sent and their respective timers set. Node9 was the first to reply indicating that it was overloaded. Likewise, node13's reply was similar.

```
Node8 GMT 499.650         TripT0.258    TIMER:PRB_TOUT    Etime: 499.908
Node8 GMT 499.651         TripT0.256    TIMER:PRB_TOUT    Etime: 499.907

Node8 receives Packet N11-N8::    Node8::Time:499.702155    Node11's reply (prc6391) was -1
node 8 sends process6391 [load = 7] to node 11
```

A further two processes arrive resulting in the random selection and probing of another two remote sites. The reply of Node11 indicates spare capacity, and process 6391 is migrated to it. In addition, Node11 becomes a member of the communicating set for Node8.

```
Node8 receives Packet N14-N8::    Node8::Time:499.767788    Node14's reply (prc6392) was 4
Node8 GMT 499.769         TripT0.256    TIMER:PRB_TOUT    Etime: 500.025

Node8 receives Packet N6-N8::     Node8::Time:499.872607    Node6's reply (prc6393) was 3
Node8 GMT 499.874         TripT0.258    TIMER:PRB_TOUT    Etime: 500.132

Node8 receives Packet N7-N8::     Node8::Time:499.925016    Node7's reply (prc6387) was 4
Node8 receives Packet N11-N8::    Node8::Time:499.926021    Node11's reply (prc6392) was 3
```

A further two new processes are created, but node11 will be probed on behalf of the first process, removing its entry until a reply is received. The second process created must therefore select a site at random as the communicating set is now empty. The replies arriving from nodes 6, 7, and 11 indicated that they too were overloaded. Only the reply for the second process is outstanding.

In terms of their load state after 4000 seconds of simulated time, the following output was produced:

N:	0	Mg:1181.00	L:	3	Rt: 2.50	Tx:	15	Imm: 989	Dth:	3274
N:	1	Mg:1251.00	L:	3	Rt: 2.43	Tx:	18	Imm:1289	Dth:	3258
N:	2	Mg:1213.00	L:	2	Rt: 2.42	Tx:	17	Imm:1263	Dth:	3200
N:	3	Mg:1161.00	L:	4	Rt: 2.64	Tx:	14	Imm:1183	Dth:	3179
N:	4	Mg:1229.00	L:	4	Rt: 2.45	Tx:	20	Imm:1216	Dth:	3214
N:	5	Mg:1277.00	L:	2	Rt: 2.41	Tx:	27	Imm:1284	Dth:	3272
N:	6	Mg:1227.00	L:	2	Rt: 2.38	Tx:	26	Imm:1282	Dth:	3207
N:	7	Mg:1269.00	L:	3	Rt: 2.49	Tx:	23	Imm:1279	Dth:	3197
N:	8	Mg:1279.00	L:	2	Rt: 2.47	Tx:	22	Imm:1325	Dth:	3184
N:	9	Mg:1249.00	L:	3	Rt: 2.43	Tx:	37	Imm:1317	Dth:	3229
N:	10	Mg:1223.00	L:	2	Rt: 2.36	Tx:	38	Imm:1314	Dth:	3136
N:	11	Mg:1151.00	L:	2	Rt: 2.44	Tx:	30	Imm:1237	Dth:	3091
N:	12	Mg:1161.00	L:	2	Rt: 2.52	Tx:	18	Imm:1156	Dth:	3236
N:	13	Mg:1225.00	L:	3	Rt: 2.66	Tx:	38	Imm:1103	Dth:	3190
N:	14	Mg:1269.00	L:	3	Rt: 2.54	Tx:	34	Imm:1210	Dth:	3225
N:	15	Mg:1250.00	L:	2	Rt: 2.63	Tx:	21	Imm:1168	Dth:	3215

In terms of their load state after 6500 seconds of simulated time, the following output was produced:

6500.00 1.5322 3.0541 4.1666 2.4466 2.29 0.3023 7.843623.80

N:	0	Mg:1776.00	L:	4	Rt: 2.49	Tx:	15	Imm:1651	Dth:	5197
N:	1	Mg:2006.00	L:	4	Rt: 2.47	Tx:	19	Imm:2001	Dth:	5253
N:	2	Mg:2028.00	L:	4	Rt: 2.50	Tx:	18	Imm:1979	Dth:	5323
N:	3	Mg:1870.00	L:	2	Rt: 2.65	Tx:	14	Imm:1915	Dth:	5132
N:	4	Mg:1940.00	L:	1	Rt: 2.47	Tx:	20	Imm:1997	Dth:	5161
N:	5	Mg:2024.00	L:	4	Rt: 2.42	Tx:	27	Imm:2099	Dth:	5268
N:	6	Mg:2029.00	L:	3	Rt: 2.46	Tx:	27	Imm:1969	Dth:	5272
N:	7	Mg:2002.00	L:	2	Rt: 2.52	Tx:	23	Imm:2040	Dth:	5192
N:	8	Mg:2015.00	L:	4	Rt: 2.52	Tx:	23	Imm:2041	Dth:	5194
N:	9	Mg:1963.00	L:	2	Rt: 2.45	Tx:	38	Imm:2089	Dth:	5178
N:	10	Mg:1963.00	L:	3	Rt: 2.32	Tx:	37	Imm:2181	Dth:	5041
N:	11	Mg:1906.00	L:	2	Rt: 2.47	Tx:	31	Imm:1961	Dth:	5124
N:	12	Mg:1899.00	L:	3	Rt: 2.56	Tx:	19	Imm:1840	Dth:	5266
N:	13	Mg:1978.00	L:	2	Rt: 2.65	Tx:	38	Imm:1823	Dth:	5205
N:	14	Mg:2045.00	L:	2	Rt: 2.50	Tx:	34	Imm:1986	Dth:	5257
N:	15	Mg:1995.00	L:	1	Rt: 2.61	Tx:	21	Imm:1867	Dth:	5188

On the basis of the above table, nodes *0, 8,* and *15* are overloaded and nodes *2, 11,12,* and *13* underloaded. However, the ratio of processes received and transmitted identifies nodes *5, 9,* and *10* to be underloaded and nodes *0, 8, 11, 13*, and *14* to be overloaded. As thousands of processes are being considered, a host is considered to be imbalanced if the difference in migration activity is significantly greater than fifty processes.

A.2.4 Global Average Neighbour (GsndNbor)

The nearest-neighbour implementation of the sender-initiated global average algorithm attempts maintain a common and representative average load amongst its neighbours. Given the mesh topology, a by-product of its operation is the propogation of this average load throughout the network. The following extract represent the output from the model during the first period of the simulation run.

SIMULATION no.8 STARTED ON Thu May 19 17:28:10 1994

No. of Processors =	16	Load value	= 0.80
Total No. of jobs =	350000	Ld Balancing Alg. = Global Avg Broadcast	
Threshold Level =	2.00	Distance	= 1
Convergence to < 2.00%			

TIME	VAR.	LOAD		DIFF.	RTime	%Conv		MIG.	Tx	%JOB
500.00	1.3489	2.7492	3.6720	2.0127	1.82	0.4036	5.1258	1.83		

N: 0	Mg:171.00	L: 3	Rt: 2.20	Tx: 13	Imm: 149	Dth: 431
N: 1	Mg:228.00	L: 4	Rt: 2.08	Tx: 14	Imm: 189	Dth: 433
N: 2	Mg:200.00	L: 2	Rt: 2.05	Tx: 11	Imm: 214	Dth: 393
N: 3	Mg:167.00	L: 2	Rt: 2.08	Tx: 10	Imm: 165	Dth: 402
N: 4	Mg:214.00	L: 4	Rt: 2.11	Tx: 13	Imm: 202	Dth: 402
N: 5	Mg:226.00	L: 2	Rt: 1.94	Tx: 13	Imm: 243	Dth: 388
N: 6	Mg:196.00	L: 1	Rt: 1.93	Tx: 13	Imm: 228	Dth: 377
N: 7	Mg:218.00	L: 3	Rt: 1.95	Tx: 11	Imm: 208	Dth: 400
N: 8	Mg:179.00	L: 5	Rt: 2.00	Tx: 11	Imm: 221	Dth: 361
N: 9	Mg:226.00	L: 2	Rt: 1.88	Tx: 13	Imm: 237	Dth: 390
N: 10	Mg:243.00	L: 1	Rt: 2.01	Tx: 14	Imm: 221	Dth: 413
N: 11	Mg:202.00	L: 2	Rt: 2.07	Tx: 11	Imm: 227	Dth: 378
N: 12	Mg:190.00	L: 3	Rt: 2.07	Tx: 11	Imm: 160	Dth: 417
N: 13	Mg:199.00	L: 3	Rt: 2.02	Tx: 11	Imm: 219	Dth: 370
N: 14	Mg:202.00	L: 4	Rt: 2.17	Tx: 13	Imm: 196	Dth: 388
N: 15	Mg:168.00	L: 3	Rt: 2.27	Tx: 11	Imm: 150	Dth: 410

Figure A.8 shows the average response time per host. The nodes offering the best average run time performance were nodes *5, 6, 7,* and *9*. The actual load state of the hosts after 500 seconds and displayed in Figure A.9, would indicate the overloaded hosts to be nodes *1, 3, 7, 8*, and *13*. Likewise, the underloaded hosts were nodes *4, 10, 11*, and *12* with process queue length of two processes or less. Only node 10 had a response time that was well below the average.

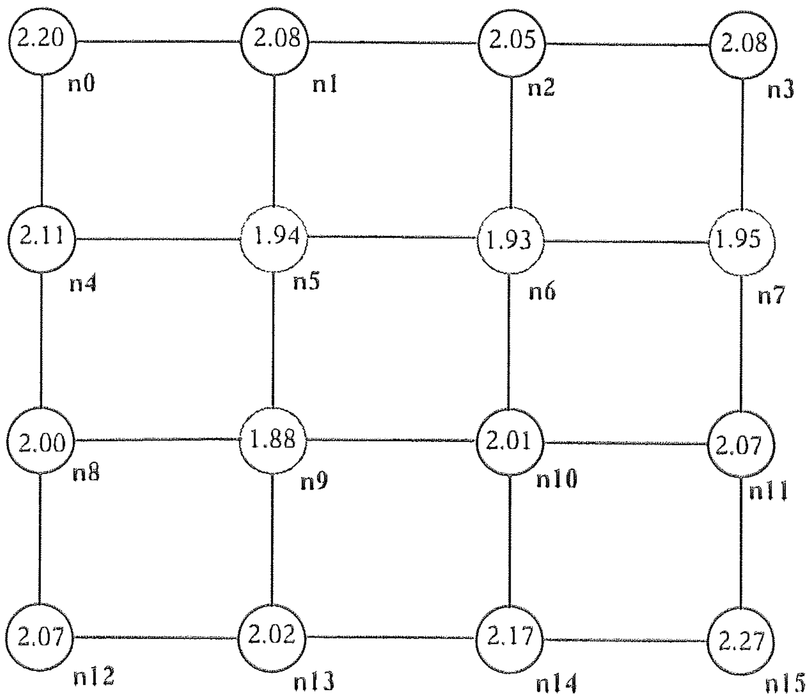


Figure A.8 Response Times After 500 Seconds using Global Average Policy

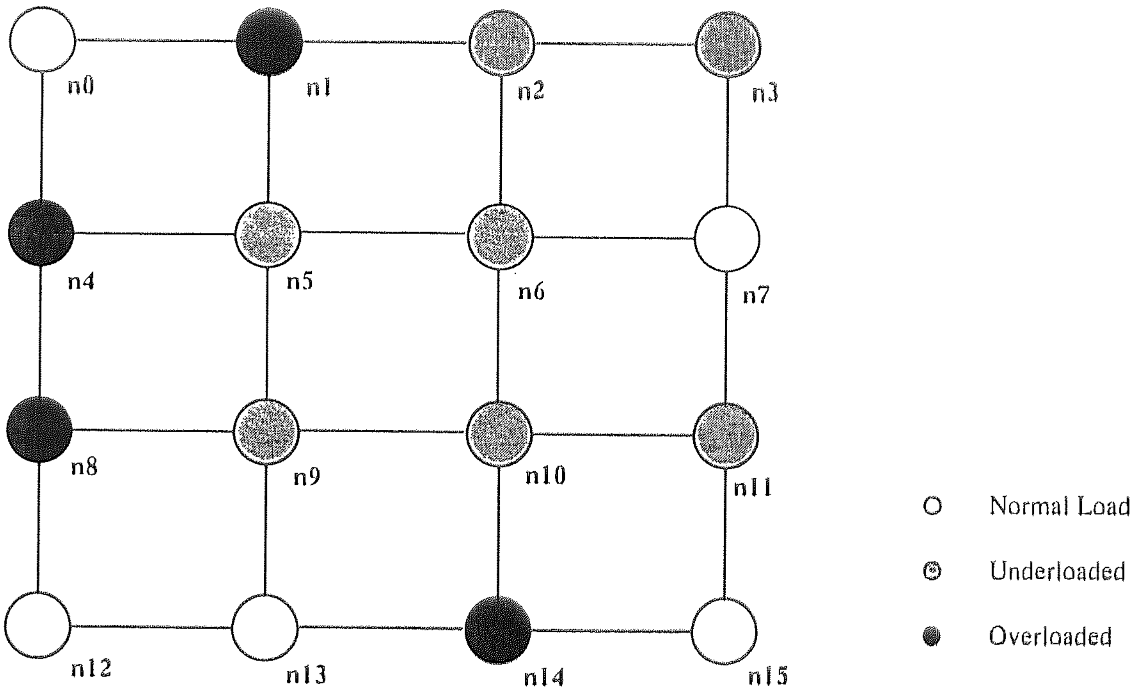
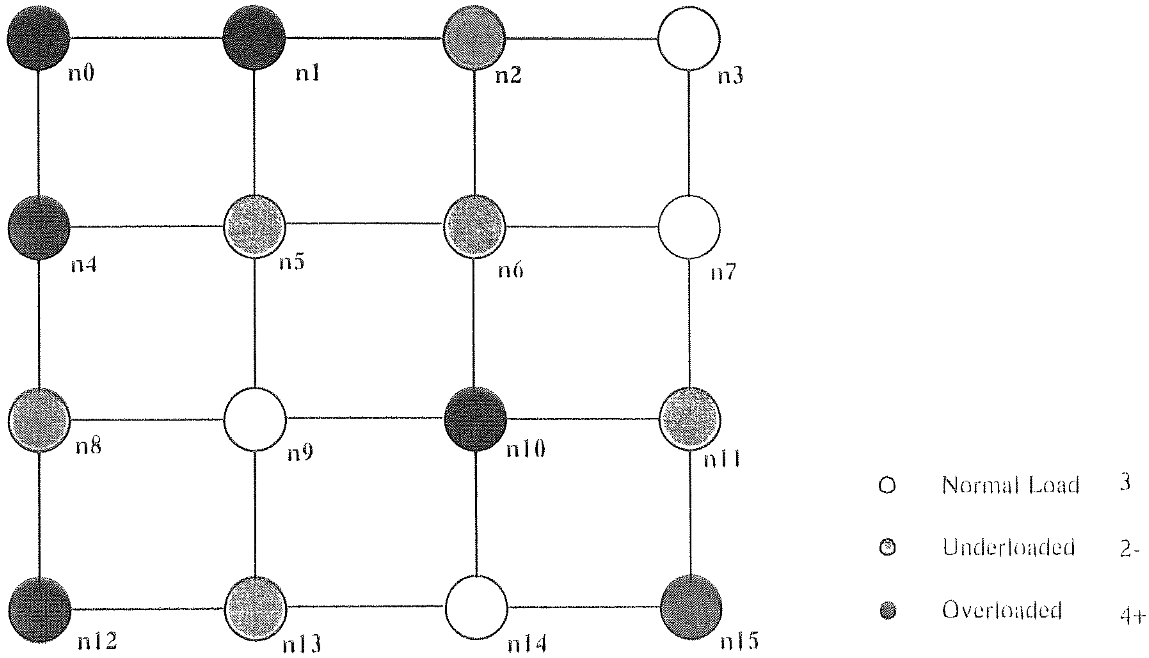


Figure A.9 Workload Distribution After 500 Seconds (Global Policy)



normal [P(rcv) - P(snd) < 10]

Overloaded [P(snd) - P(rcv) > 10]

Underloaded [P(rcv) - P(snd) > 10]

Figure A.10 Workload Distribution After 500 Seconds (Tx/Rx Ratio)

In Figure A.10, based on the ratio of processes sent and received over time, nodes *0, 1, 4, 10, 12, 15* would appear to be the overloaded nodes, whilst nodes *2, 5, 6, 8, 11,* and *13* are underloaded.

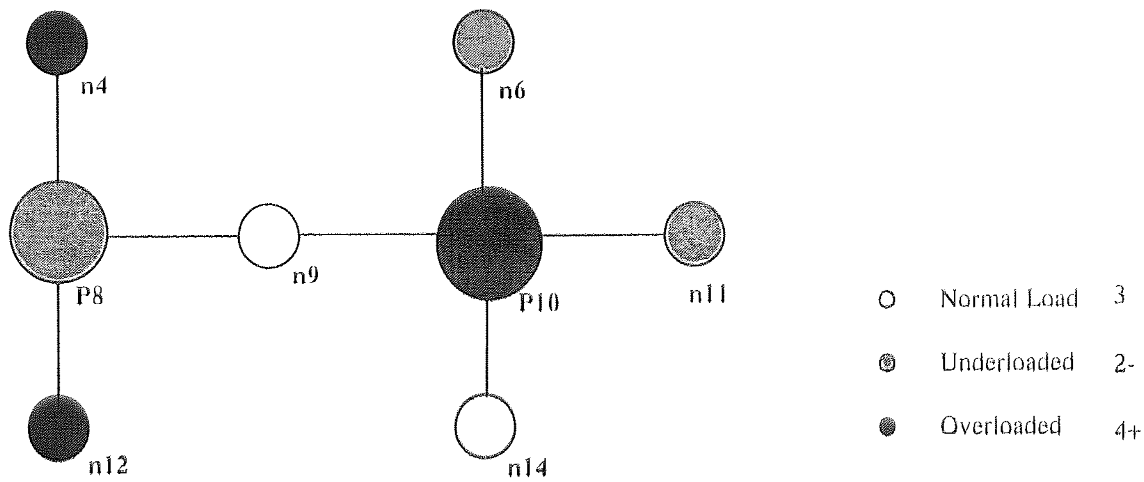


Figure A.11 Nearest-Neighbour Network Partition: Perspective of nodes 10 and 8

Consider the nodes 8 and 10 where the former is underloaded and the latter overloaded. Node8 has three neighbours, two of which are overloaded. The other neighbour, node9 it has in common with node10. In contrast node 10 has four neighbours two of which are underloaded and two with normal loads. Figure A.11 shows the relationship between node 8 and 10 and their most immediate neighbours. The node (node9) common to both p8 and p10 is likely to be the point for the indirect propagation of the load average between the sites concerned.

The following trace focussed on the message dialogue of node*10* and node*8* just before and after the first period of the simulation run.

```
Node8 receives Packet N9-N8::     node 8 receives Exit msg [prc:: 6332 at node:9]
Node8 receives Packet N12-N8::    node 8 receives NewThreshold:2 [load:: 2] from node:12

Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept
Node8 GMT 496.775      TripT0.252    TIMER:WAITP_TOUT       Etime: 497.027

Node8 receives Packet N4-N8::     node 8 receives NewThreshold:2 [load:: 2] from node:4

Node8 receives Packet N12-N8::    node 8 receives Exit msg [prc:: 6326 at node:12]

WAITP TIMED OUT for Node8 at 497.030
```

Node8 receives exit messages for processes 6332 and 6326 from node9 and 12 respectively. It also received new load messages from node 12 and 4, setting the threshold level to two. As node8 is underloaded, a high load message from node12 resulted in the setting of the "await process" timer. Eventually, it times out as node12 may have chosen to migrate the process to one of its earlier respondents.

```
Node8 GMT 497.230      TripT0.252    TIMER:LOW_TOUT   Etime: 497.482
Node10 GMT 497.386     TripT0.252    TIMER:LOW_TOUT   Etime: 497.638

LOWLD TIMED OUT for Node8 at 497.493        NewTh= 1

Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept
Node8 GMT 497.504      TripT0.252    TIMER:WAITP_TOUT       Etime: 497.756

LOWLD TIMED OUT for Node10 at 497.646       NewTh= 1

Node10 GMT 497.646     TripT0.252    TIMER:LOW_TOUT   Etime: 497.898

Node10 receives Packet N11-N10::  Node11 -overloaded- has asked node10 TO accept
Node10 GMT 497.677     TripT0.252    TIMER:WAITP_TOUT       Etime: 497.929

Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept

WAITP TIMED OUT for Node8 at 497.767
WAITP TIMED OUT for Node10 at 497.938
```

Both nodes 8 and 10 are underloaded resulting in the setting of their respective "low load" timers. As node 8 had not received an enquiry from an overloaded host it eventually times out, assume the load threshold to be too high, and decrements and broadcast a new threshold value of one. As node12 is an immediate neighbour, it now becomes overloaded and sends an enquiry to all its neighbours (including node8). A similar pattern of behaviour is exhibited by node10. However, both sites were not selected as recipient sites and subsequently timed out.

```
Node8 GMT 497.987        TripT0.252    TIMER:LOW_TOUT  Etime: 498.239
Node8 receives Packet N12-N8::    node 8 receives NewThreshold:2 [load:: 1] from node:12

Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept
Node8 GMT 498.019        TripT0.252    TIMER:WAITP_TOUT      Etime: 498.271
Node8 GMT 498.240        TripT0.252    TIMER:LOW_TOUT  Etime: 498.492

Node8 receives Packet N12-N8::    Node12 -overloaded- has asked node8 TO accept
Node8 GMT 498.281        TripT0.252    TIMER:WAITP_TOUT      Etime: 498.533

Node8 receives Packet N12-N8::
node 8 RECEIVES process6365 [load = 3] FROM node 12
```

As node 8 is still underloaded, it sets "low load" timer, but receives a new threshold value of two from node12. Consecutive enquiries by node12 resulted in the migration of process 6365 to node8.

```
Node10 receives Packet N14-N10::  Node14 -overloaded- has asked node10 TO accept
Node8 receives Packet N4-N8::     Node4 -overloaded- has asked node8 TO accept

WAITP TIMED OUT for Node8 at  498.568
Node8 receives Packet N4-N8::     node 8 receives Exit msg [prc:: 6344 at node:4]

Node8 receives Packet N9-N8::     Node9 -overloaded- has asked node8 TO accept
Node8 GMT 498.721        TripT0.252    TIMER:WAITP_TOUT      Etime: 498.973

Node10 receives Packet N9-N10::   Node9 -overloaded- has asked node10 TO accept
Node10 receives Packet N14-N10::  Node14 -overloaded- has asked node10 TO accept

Node8 receives Packet N9-N8::     Node9 -overloaded- has asked node8 TO accept

WAITP TIMED OUT for Node8 at  498.976
```

Node 10 has received enquiries from nodes 14 and 9. These are rejected as there is no spare capacity on node10. Node 8 also received enquiries in sequence from nodes 4 and 9, but only the latter invoked a "wait for process" timer. However, node 9 decides not to send the process to node8 and it therefore times out.

```
Node10 receives Packet N14-N10::  node 10 receives NewThreshold:2 [load:: 2] from node:14

Node10 receives Packet N9-N10::   Node9 -overloaded- has asked node10 TO accept
Node10 GMT 498.962       TripT0.252    TIMER:WAITP_TOUT      Etime: 499.214
Node10 receives Packet N9-N10::
```

node 10 RECEIVES process6370 [load = 3] FROM node 9

Node8 GMT 499.027 TripT0.252 TIMER:LOW_TOUT Etime: 499.279
LOWLD TIMED OUT for Node8 at 499.329 NewTh= 2

Node10 GMT 499.430 TripT0.252 TIMER:LOW_TOUT Etime: 499.682
LOWLD TIMED OUT for Node10 at 499.732 NewTh= 2

Node 10 receives a new threshold value of two from node14. An enquiry is received from node9 resulting in the subsequent migration of process 6370 to node10. Nodes 8 and 10 become underloaded, set their respective "low load" timer, and both time out as a result of the absence of any enquiries by overloaded sites. The current threshold of three is assumed to be too high and is therefore decremented and broadcast to neighbouring sites.

Node8 GMT 499.787 TripT0.252 TIMER:HIGH_TOUT Etime: 500.039

Node10 receives Packet N9-N10::
node 10 receives NewThreshold:2 [load:: 2] from node:9

Node8 receives Packet N9-N8::
node 8 receives NewThreshold:2 [load:: 5] from node:9

Node8 receives Packet N4-N8:: (4)
Node8 receives Packet N12-N8::(2)
Node8 receives Packet N9-N8:: (2)

Node10 GMT 499.936 TripT0.252 TIMER:LOW_TOUT Etime: 500.188

Node8 is overloaded, sets the "high load" timer and broadcasts an implicit request for assistance to its neighbours. Both node8 and 9 receive a new load message from node 9 which results in the cancellation of any load timers. The replies received by node8 from nodes 4, 12, and9 to its "high load" enquiry would indicate that both 12 and 9 had spare capacity. However, in light of the earlier change in the threshold all three respondents are considered as being overloaded.

500.00 1.3489 2.7492 3.6720 2.0127 1.82 0.4036 5.1258 1.83

Node8 GMT 500.046 TripT0.252 TIMER:HIGH_TOUT Etime: 500.298
Node8 receives Packet N4-N8:: (4)
Node8 receives Packet N12-N8:: (4)
Node8 receives Packet N9-N8:: (3)

LOWLD TIMED OUT for Node10 at 500.200 NewTh= 1

Node10 receives Packet N14-N10:: Node14 -overloaded- has asked node10 TO accept
Node10 GMT 500.241 TripT0.252 TIMER:WAITP_TOUT Etime: 500.493

HIGHLD TIMED OUT for Node8 at 500.304 NewTh= 3
Node8 GMT 500.307 TripT0.252 TIMER:HIGH_TOUT Etime: 500.559

Node10 receives Packet N14-N10::
node 10 RECEIVES process6387 [load = 2] FROM node 14

Node8 receives Packet N12-N8::(2) node 8 sends process6398 [load = 6] to node 12
Node8 receives Packet N9-N8::(2)
Node8 receives Packet N4-N8::(3)

As node8 is overloaded a "high load" message is broadcast to its neighbours. However, the replies indicate that nodes 4, 12 , and 9 are overloaded. Therefore, node 8 times out, assumes the threshold to be too low, and increments and broadcast a new threshold value of three. When the algorithm is next executed, node8 is still overloaded, but two of its neighbours (nodes 12 and 9) are underloaded. Process 6387 is migrated to the first respondent, node12. In contrast, an underloaded node 10 assume the threshold to be too high, decrements and broadcast a threshold value of one which results in the migration of process 6387 from node14.

Node8 GMT 500.578 TripT0.252 TIMER:HIGH_TOUT Etime: 500.830

Node10 receives Packet N14-N10:: Node14 -overloaded- has asked node10 TO accept
Node10 receives Packet N6-N10:: Node6 -overloaded- has asked node10 TO accept
Node8 receives Packet N4-N8:: Node4 -overloaded- has asked node8 TO accept

Node8 receives Packet N12-N8::(3) node 8 sends process6392 [load = 5] to node 12

Node8 receives Packet N9-N8:: (2)
Node8 receives Packet N4-N8::(3)

Node8 receives Packet N4-N8:: node 8 receives NewThreshold:2 [load:: 4] from node:4
Node8 GMT 500.851 TripT0.252 TIMER:HIGH_TOUT Etime: 501.103

Node10 receives Packet N6-N10:: node 10 receives NewThreshold:2 [load:: 2] from node:6

Node 8 is overloaded and broadcast to its neighbours. The response by all neighbours indicates there load to be below the threshold, but only the first respondent is selected as recipient for the excess process load. The subsequent arrival of a new load message from neighbouring node4 would indicate that it has been unsuccessful in receiving remote processes, assumed the threshold to be too high and therefore broadcast a lower threshold value. In a similar manner, Node 10 received enquiries from overloaded sites 14 and 6. As these were rejected, a new load message duly arrived from node 6.

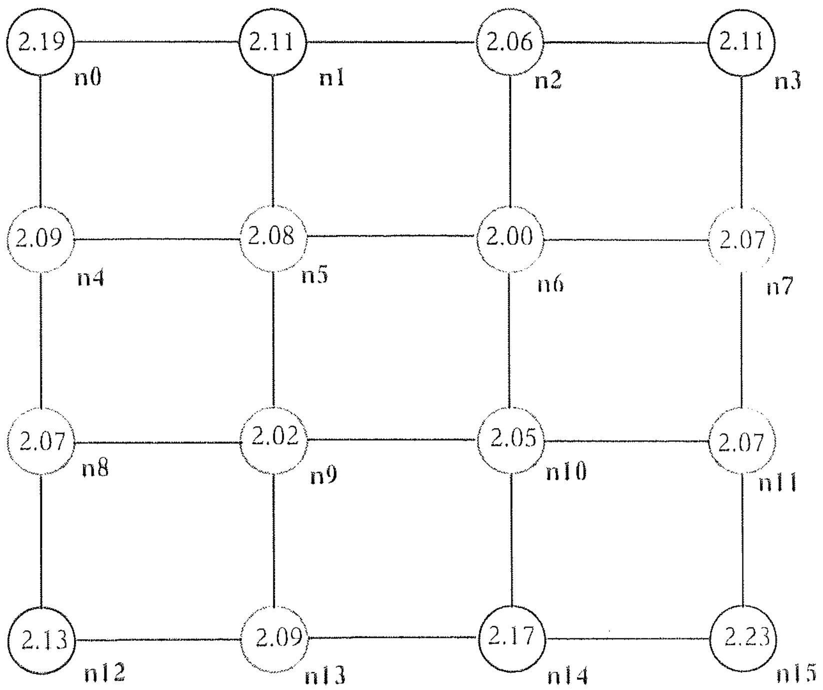
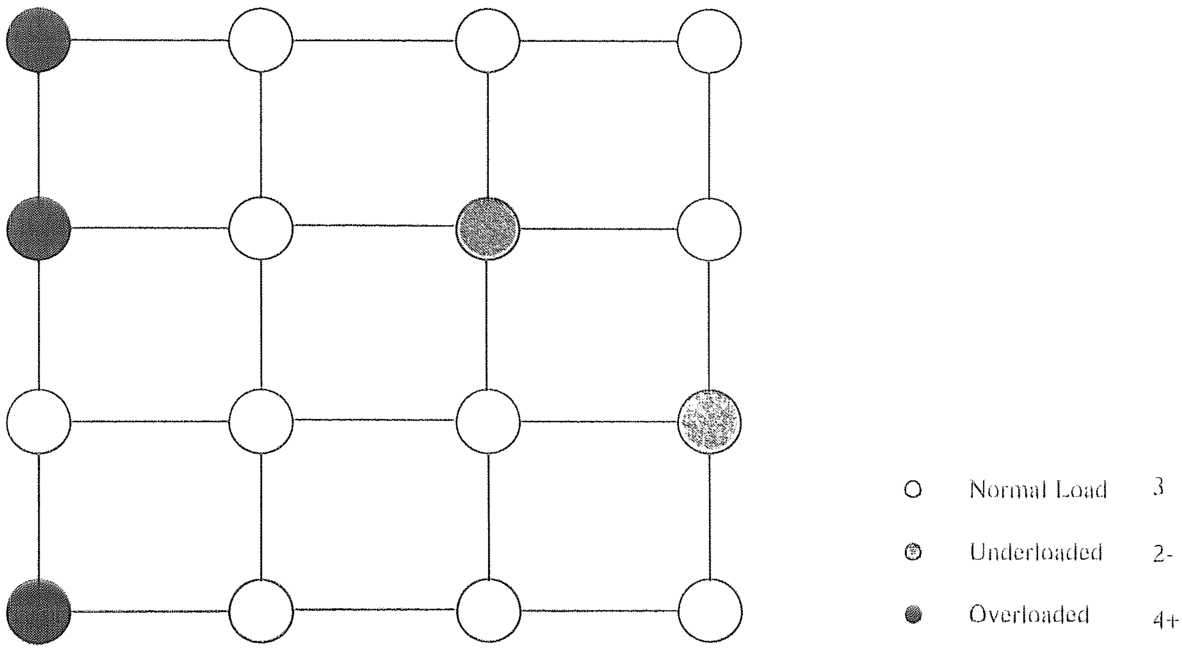


Figure A.12 Response Time After 3000 seconds (Global Average Policy)

Figure A.12 shows the runtime performance after 3000 seconds of simulation time. Whilst the average response time has increased marginally, the run-time performance of nodes 0, 4, and 15 have improved. However, nodes *6* and *9* continue to yield better than average performance.

In terms of their load state after 3000 seconds of simulated time, the following output was produced:

N: 0	Mg:963.00	L: 1	Rt: 2.19	Tx: 12	Imm: 897	Dth: 2460
N: 1	Mg:1283.00	L: 2	Rt: 2.11	Tx: 12	Imm:1242	Dth: 2440
N: 2	Mg:1274.00	L: 2	Rt: 2.06	Tx: 11	Imm:1291	Dth: 2398
N: 3	Mg:992.00	L: 1	Rt: 2.11	Tx: 11	Imm: 992	Dth: 2382
N: 4	Mg:1287.00	L: 4	Rt: 2.09	Tx: 12	Imm:1217	Dth: 2441
N: 5	Mg:1400.00	L: 3	Rt: 2.02	Tx: 13	Imm:1440	Dth: 2416
N: 6	Mg:1340.00	L: 3	Rt: 2.00	Tx: 13	Imm:1418	Dth: 2379
N: 7	Mg:1314.00	L: 3	Rt: 2.07	Tx: 12	Imm:1287	Dth: 2416
N: 8	Mg:1235.00	L: 2	Rt: 2.07	Tx: 12	Imm:1281	Dth: 2371
N: 9	Mg:1414.00	L: 4	Rt: 2.02	Tx: 13	Imm:1442	Dth: 2399
N: 10	Mg:1422.00	L: 3	Rt: 2.05	Tx: 13	Imm:1455	Dth: 2367
N: 11	Mg:1269.00	L: 1	Rt: 2.07	Tx: 11	Imm:1359	Dth: 2317
N: 12	Mg:1055.00	L: 1	Rt: 2.13	Tx: 11	Imm: 944	Dth: 2443
N: 13	Mg:1262.00	L: 3	Rt: 2.09	Tx: 12	Imm:1310	Dth: 2378
N: 14	Mg:1311.00	L: 2	Rt: 2.17	Tx: 12	Imm:1282	Dth: 2428
N: 15	Mg:1027.00	L: 3	Rt: 2.23	Tx: 11	Imm: 990	Dth: 2401



Overloaded [P(snd) - P(rcv) >> 50]

Underloaded [P(rcv) - P(snd) >>50]

Figure A.13 Workload After 3000 seconds (Global Average Policy)

The output indicate that nodes *4, and 10* are overloaded and nodes *0, 1, 2, 3, 8, 11, 12, and 14* underloaded. However, the ratio of the processes received and transmitted identifies nodes *6* and *11* to be underloaded and nodes *0, 4,* and *12,* to be overloaded. This is illustrated in Figure A.12.

APPENDIX B

PERFORMANCE STATISTICS FOR ALGORITHMS

B.1 THE QUEUING MODEL

The graphs in Figure B.1.1 to B.1.10 is representative of the exponential distribution obtained with a total of 100000 processes passing through the system. The load to the system was varied by using the formula proposed by Lavenberg [Lavenberg83] for *traffic intensity* for manipulating the mean interarrival time:

$$\varrho = \lambda E[S]/m\gamma,$$

where $\lambda = 1/E[T]$,

and the notation is explained thus:

λ	arrival rate
γ	service rate
E[S]	Expected service time
E[T]	Mean interarrival time
ϱ	system load (stable if $\varrho < 1$)

Thus, lengthening the mean interarrival time (giving a low value for ϱ such as 0.2 for example) would represent light system load, and shortening it (such as $\varrho = 0.9$) heavy system load. The arriving processes are considered to form a Poisson stream where the interarrival times are independent and identically distributed and are exponentially distributed. The uniformly distributed random number U was accomplished using the UNIX function *erand48* and *drand48*. Both functions are based on the linear congruential algorithm and 48-bit integer arithmetic. The following inverse transformation was then applied:

$$-\lambda^{-1}\log_e U,$$

to form an exponentially distributed random number with mean $1/\lambda$. By using a Poisson process, the system job stream can be split into k substreams, where each substream is Poisson with mean arrival rate λ/k, and exponentially distributed interarrival times. The charts in figures B.1.1 to B.1.10 illustrates this property for both the M/M/1 and M/M/16 queuing models using a Poisson process.

Therefore, the job stream for each host was achieved by first creating k separate data files (jobs^{0}_pg0 to jobs^{k-1}_pg0), and generating and splitting a Poisson arrival process using the function *erand48* to select the data file for recording the next arrival time. The UNIX function *erand48* was also used to create an independent distribution stream for each host in cases where the location policy of the load balancing algorithm needs to randomly select a remote site.

Whilst the results produced by a single simulation run was generally representative of the performance achievable with and without load balancing a total of 20 simulation runs, using different seeds for the pseudo random number generator functions, were conducted per experiment. This was found to be particularly important at high system loads and large processor mesh where performance patterns can exhibit significant fluctuations from one simulation run to another.

Each experiment would run until the average response time for each host differed by less than two percent. Thus, most simulation runs converged to response times of less than two percent after 3600 seconds of simulation time. However, under heavy load conditions, the differences in the performance of edge nodes compared to the more central nodes are re-enforced but stabilise over a longer period of simulated time, namely 8000 to 12000 seconds. Therefore, a 95% confidence interval criteria was used for terminating these experiments. The results produced at low to moderate system load was also found to be consistent with the criteria applied at heavy system load.

Figures B.1.11 to B.1.12, and B.1.13 to B.1.14 shows the average system workload and response times, over the first 1000 seconds of simulated time, for low and heavy system loads respectively. In all cases where load balancing is inactive, the system load and response times stabilise after 600 seconds of simulated time. It is only the global broadcast algorithm that becomes increasingly unstable under extreme load conditions.

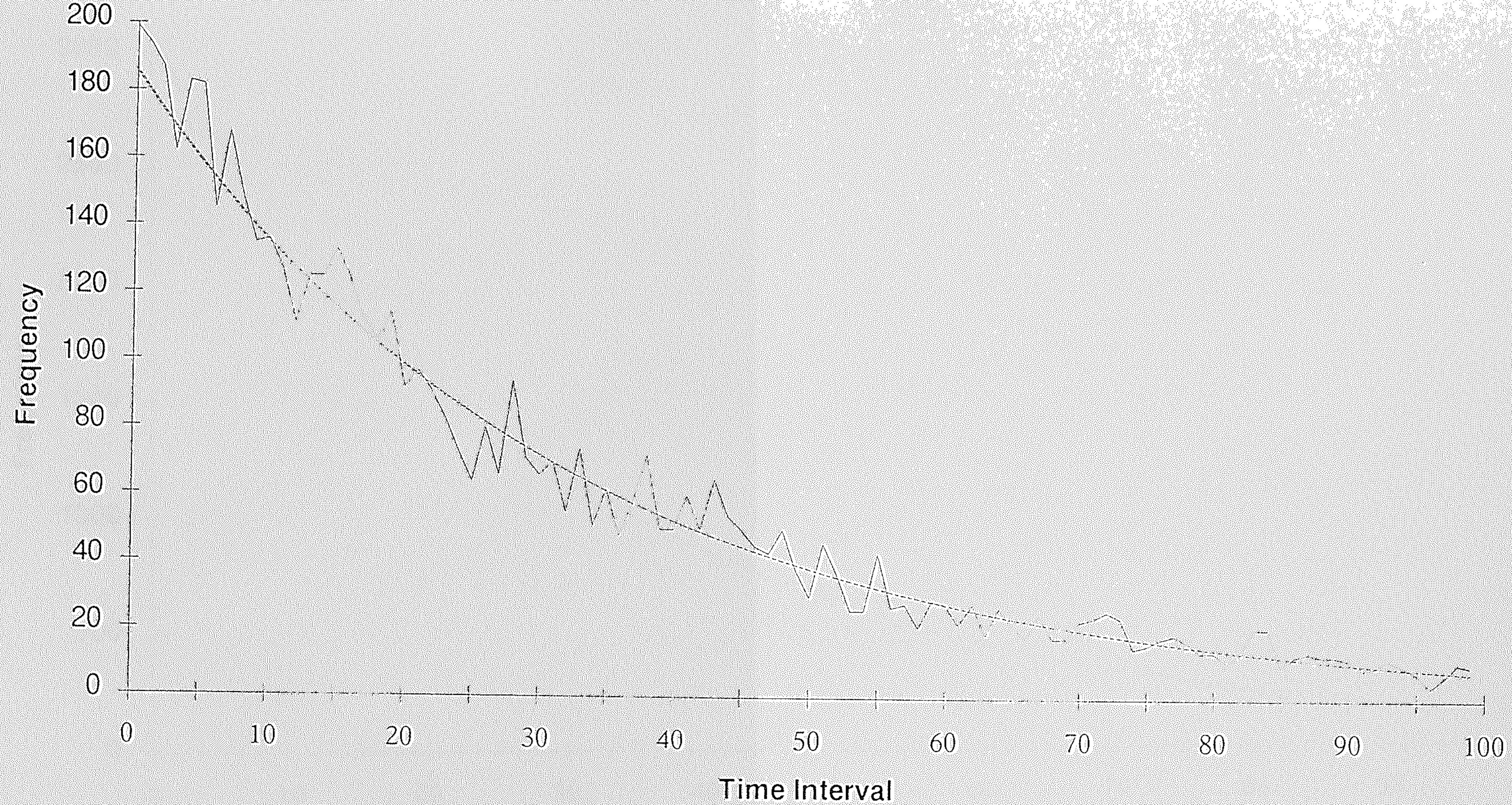


Figure B.1.1 Light Weight Processes (M/M/1) : Distribution Curve ($\rho = 0.2$)

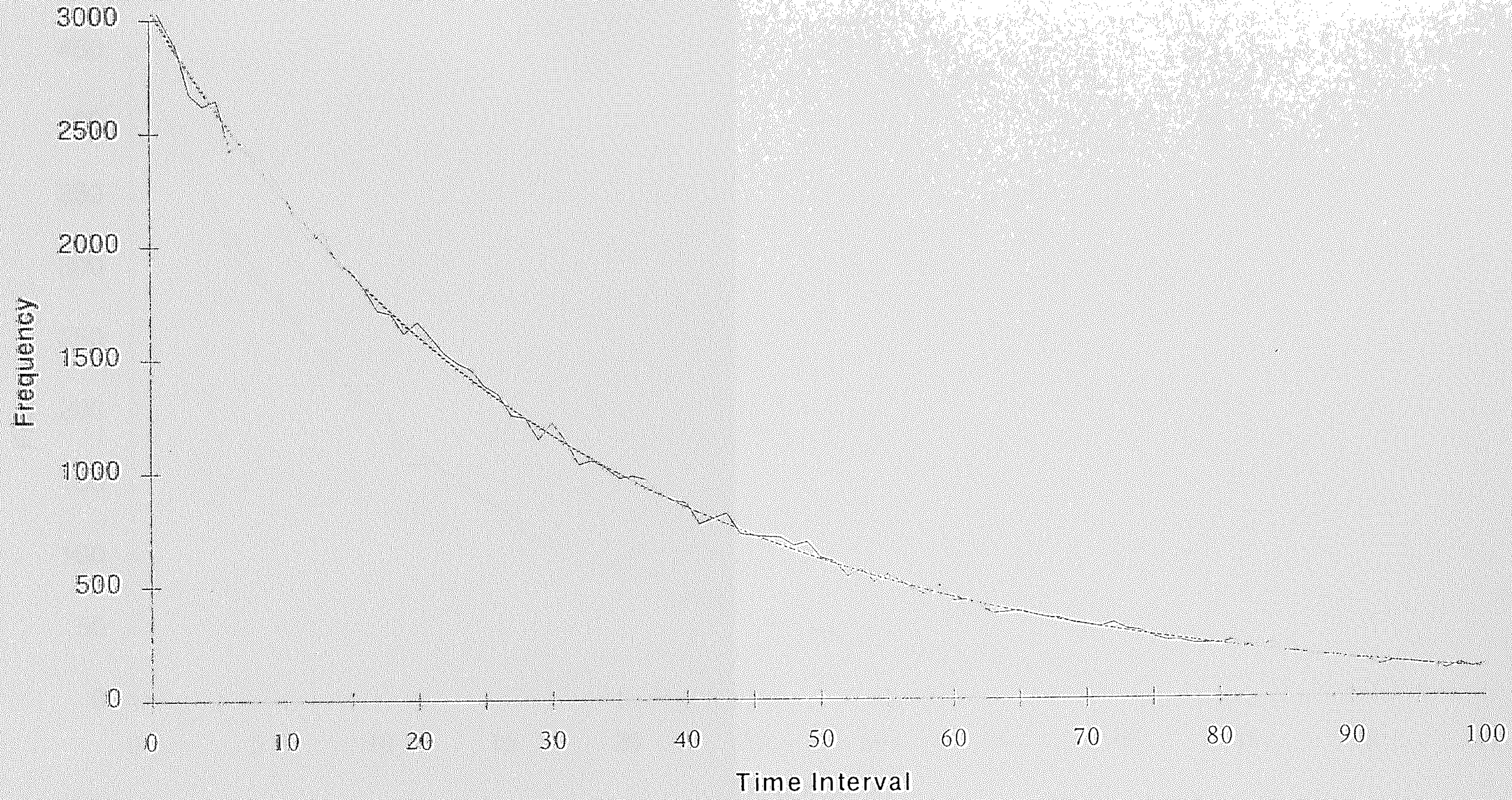


Figure B.1.2 Light Weight Processes (M/M/16) - Distribution Curve ($\varrho = 0.2$)

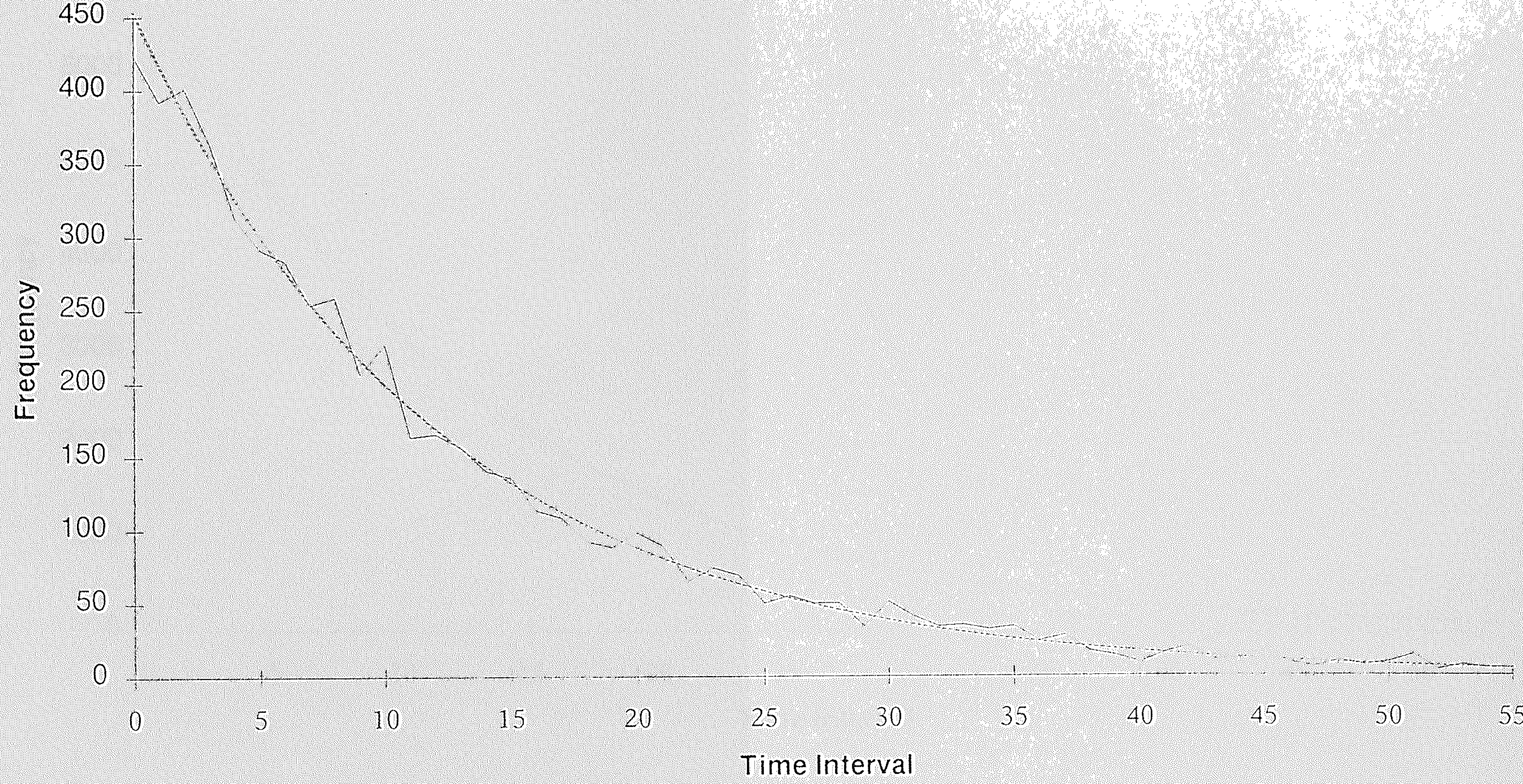


Figure B.1.3 Light Weight Processes (M/M/1) : Distribution Curve ($\varrho = 0.5$)

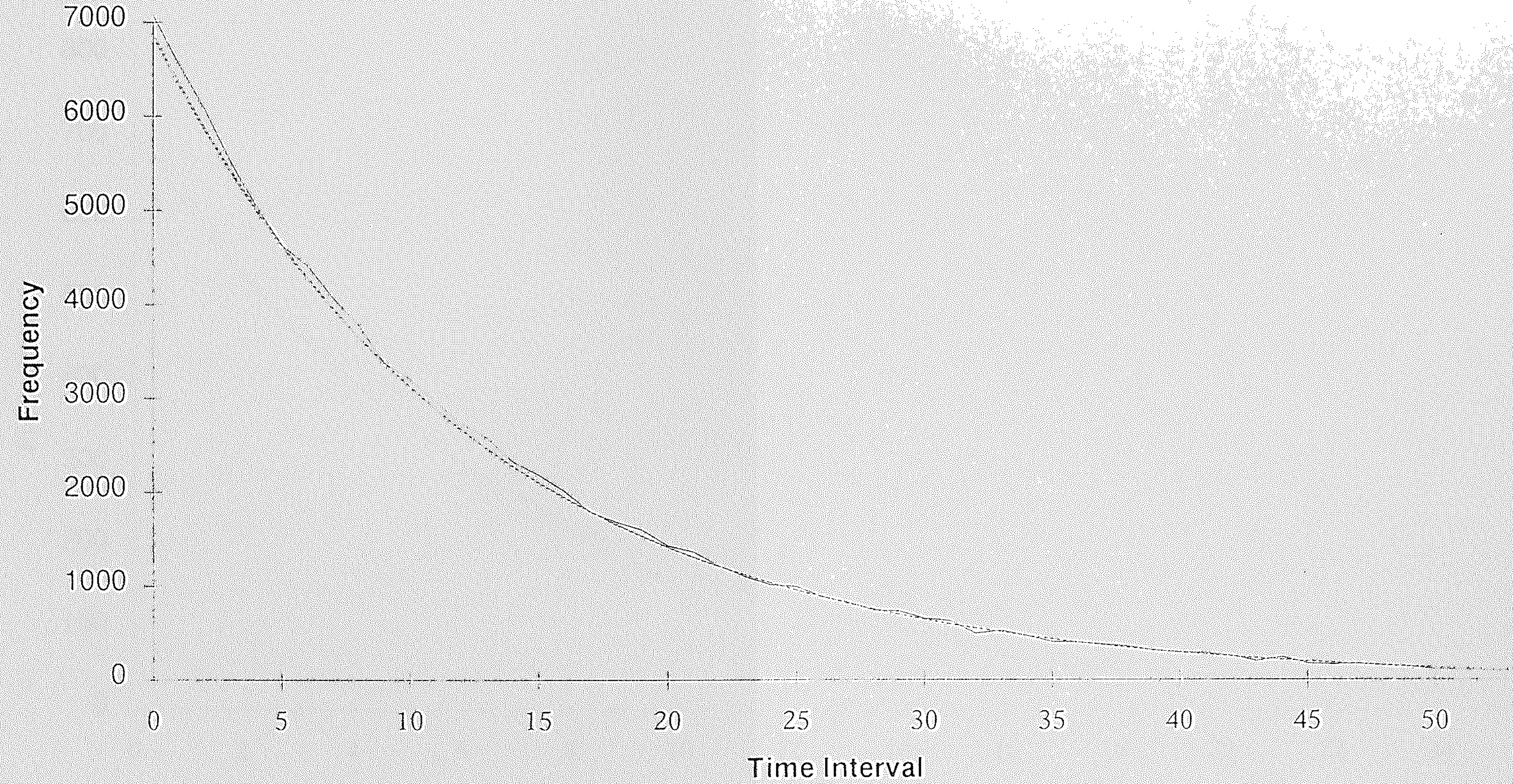


Figure B.1.4 Light Weight Processes (M/M/16) : Distribution Curve ($\varrho = 0.5$)

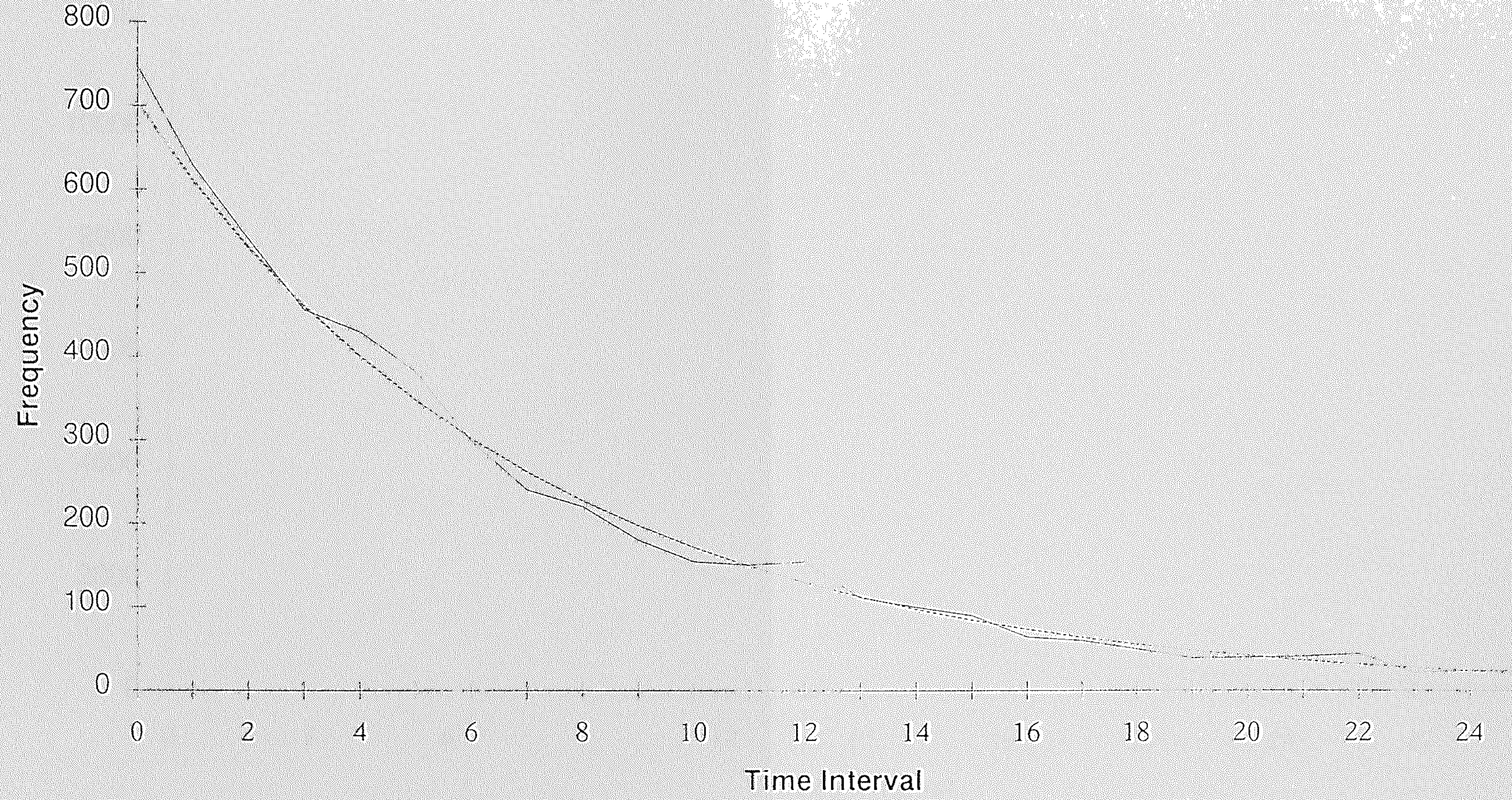


Figure B.1.5 Light Weight Processes (M/M/1) : Distribution Curve ($\varrho = 0.9$)

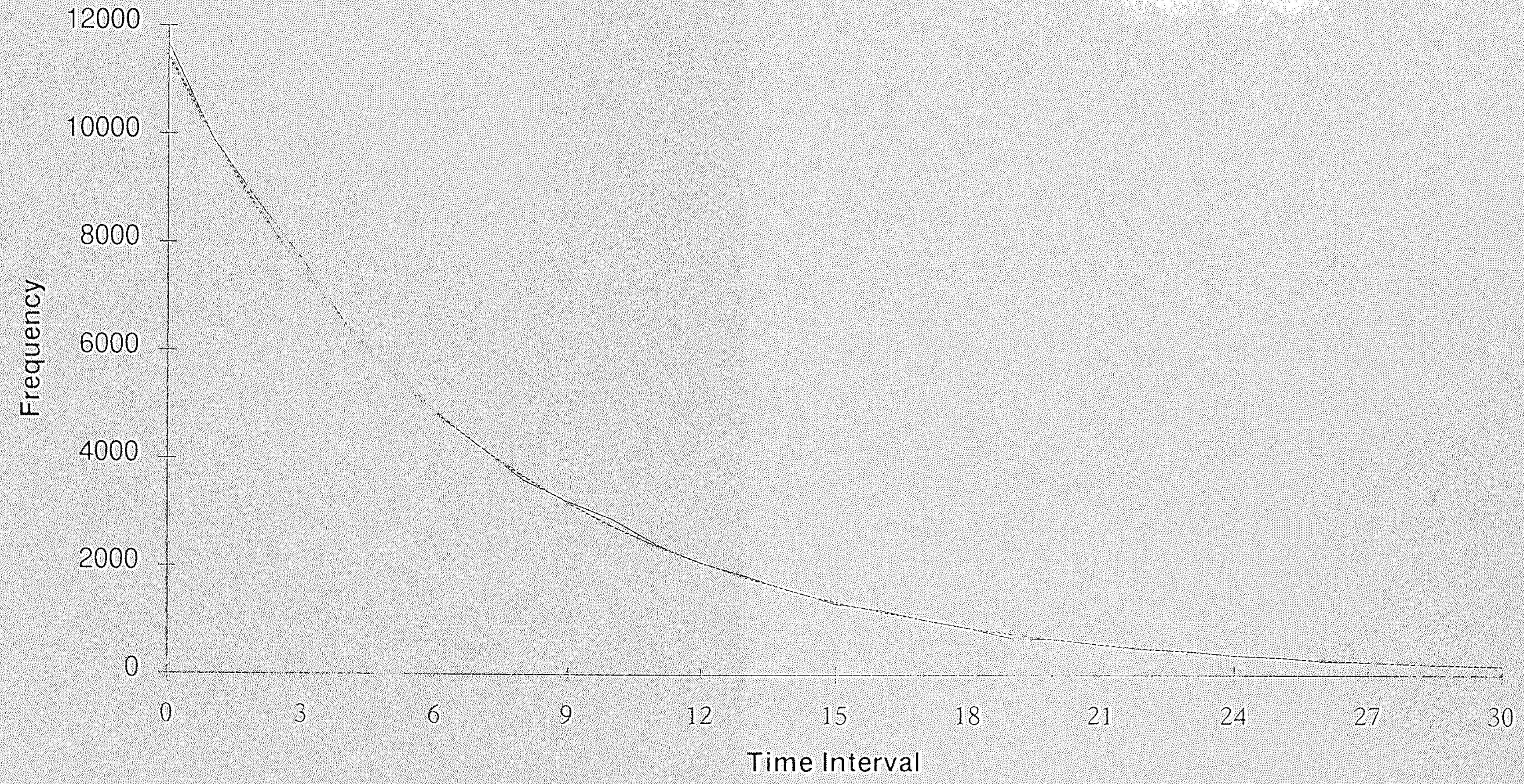


Figure B.1.6 Light Weight Processes (M/M/16) : Distribution Curve ($\varrho = 0.9$)

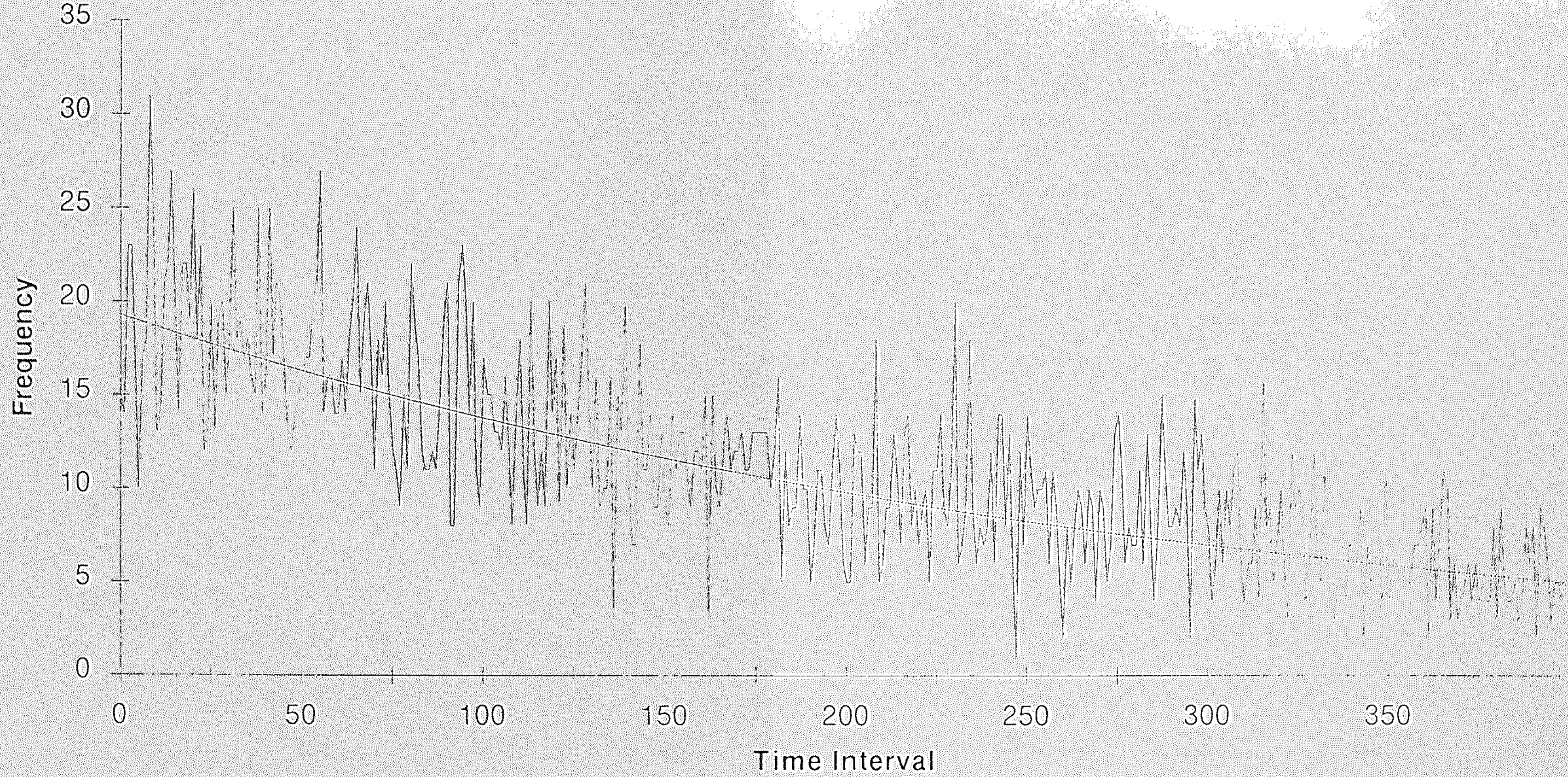


Figure B.1.7 Heavy Weight Processes (M/M/1) : Distribution Curve ($\varrho = 0.9$)

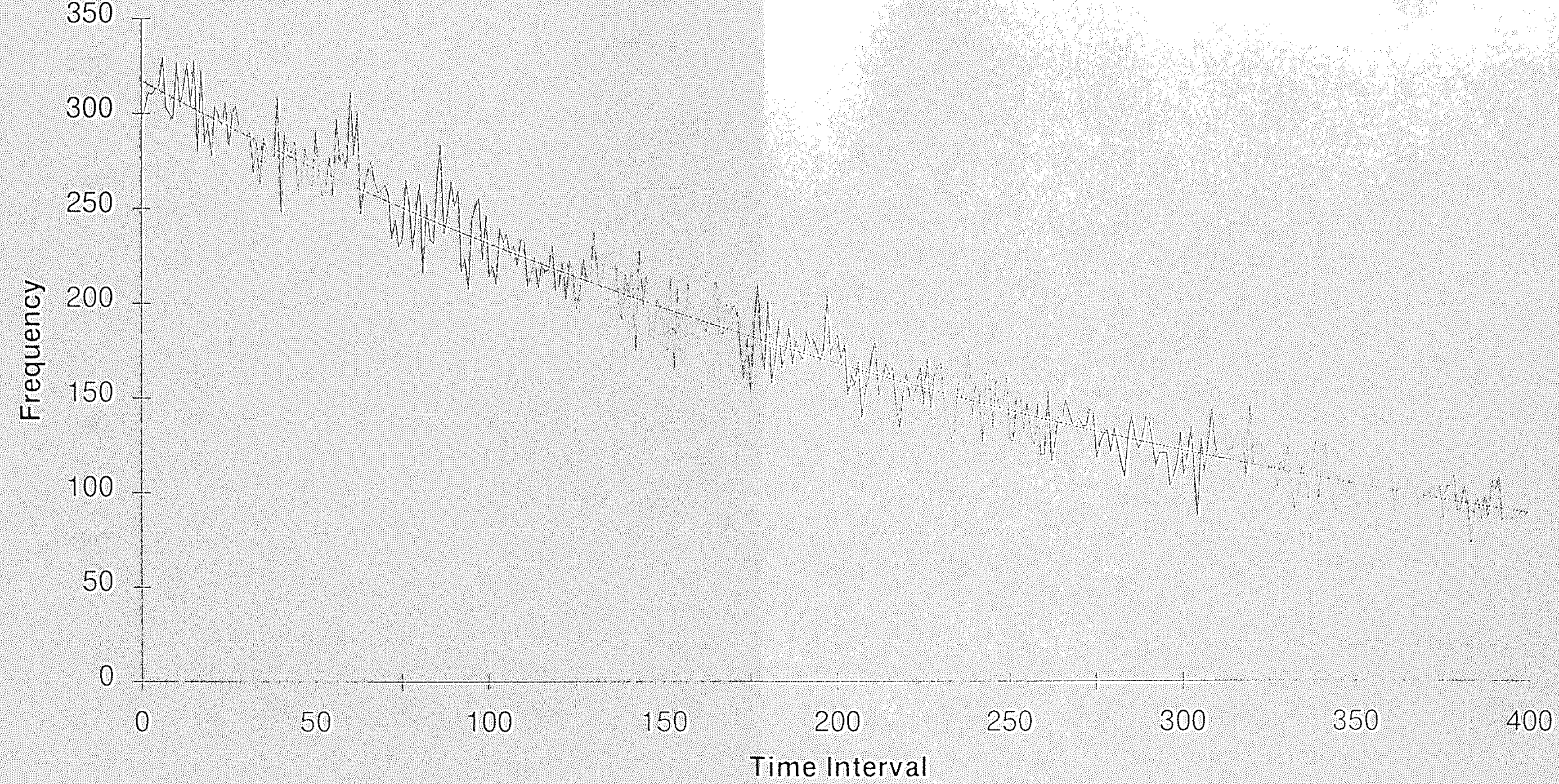


Figure B.1.8 Heavy Weight Processes (M/M/16) - Exponential Distribution Curve ($\varrho = 0.9$)

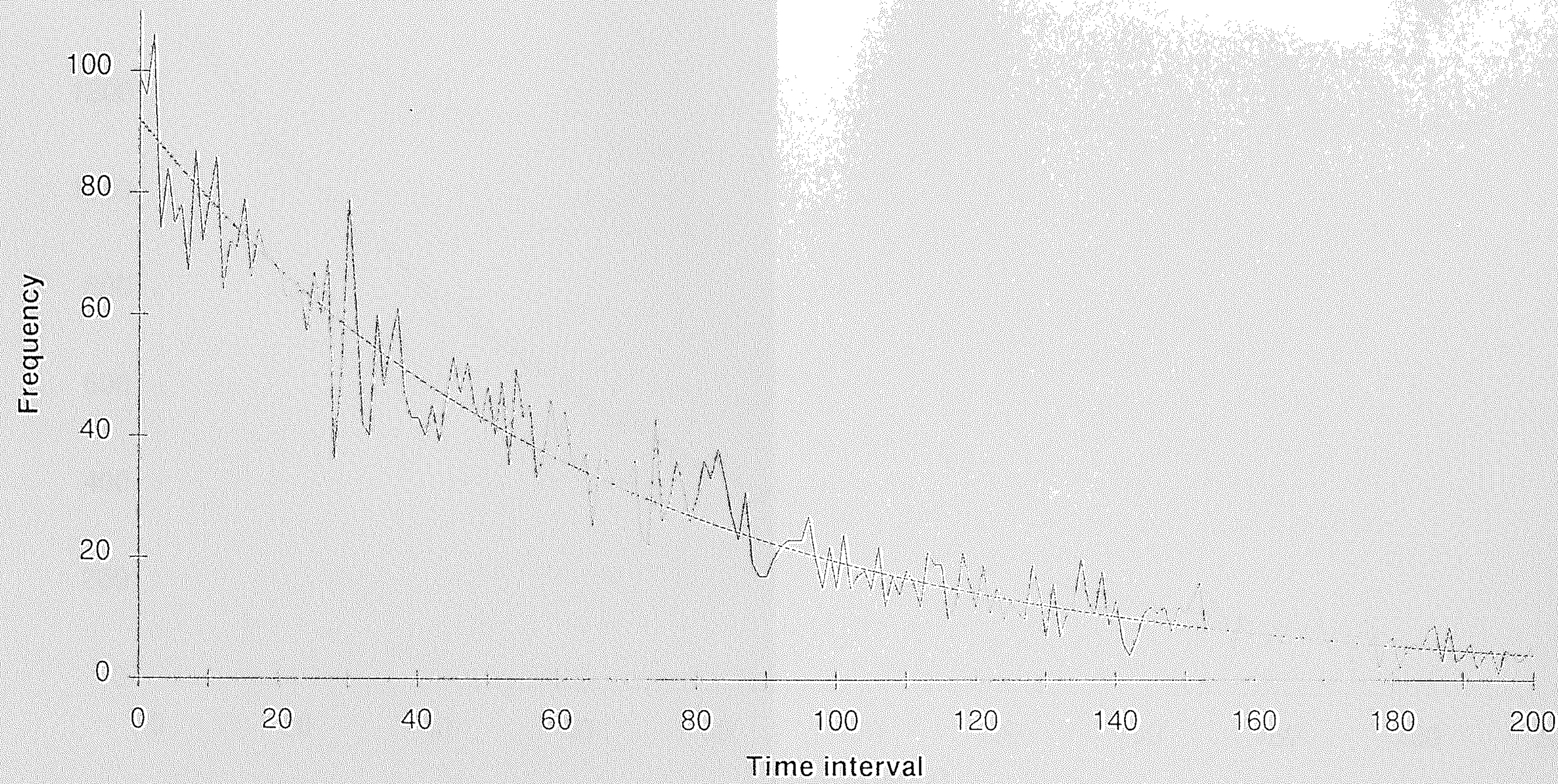


Figure B.1.9 Heavy Weight Processes (M/M/1) : Distribution Curve ($\varrho = 0.9$)

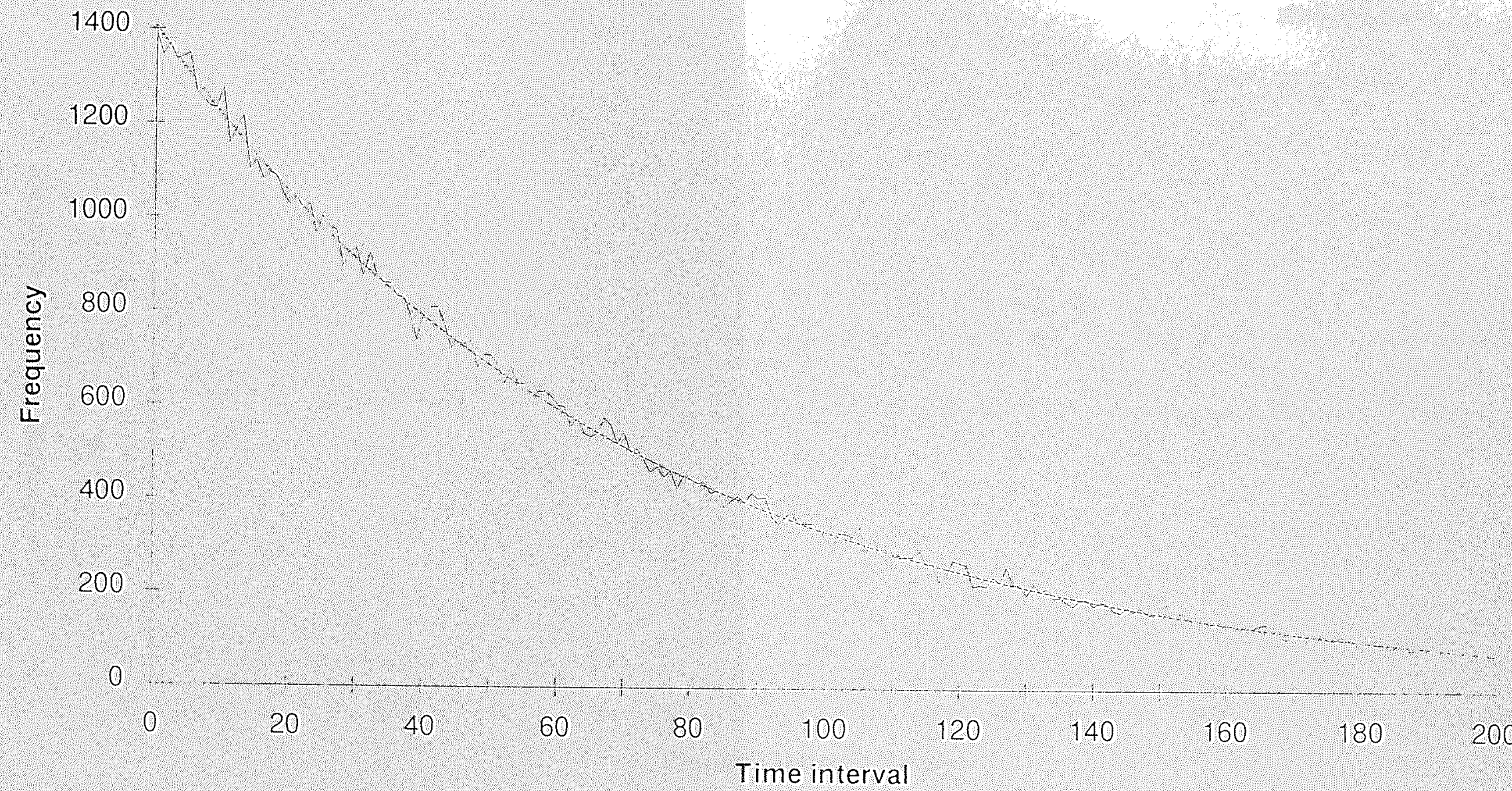


Figure B.1.10 Heavy Weight Processes (M/M/16) : Distribution Curve ($\varrho = 0.9$)

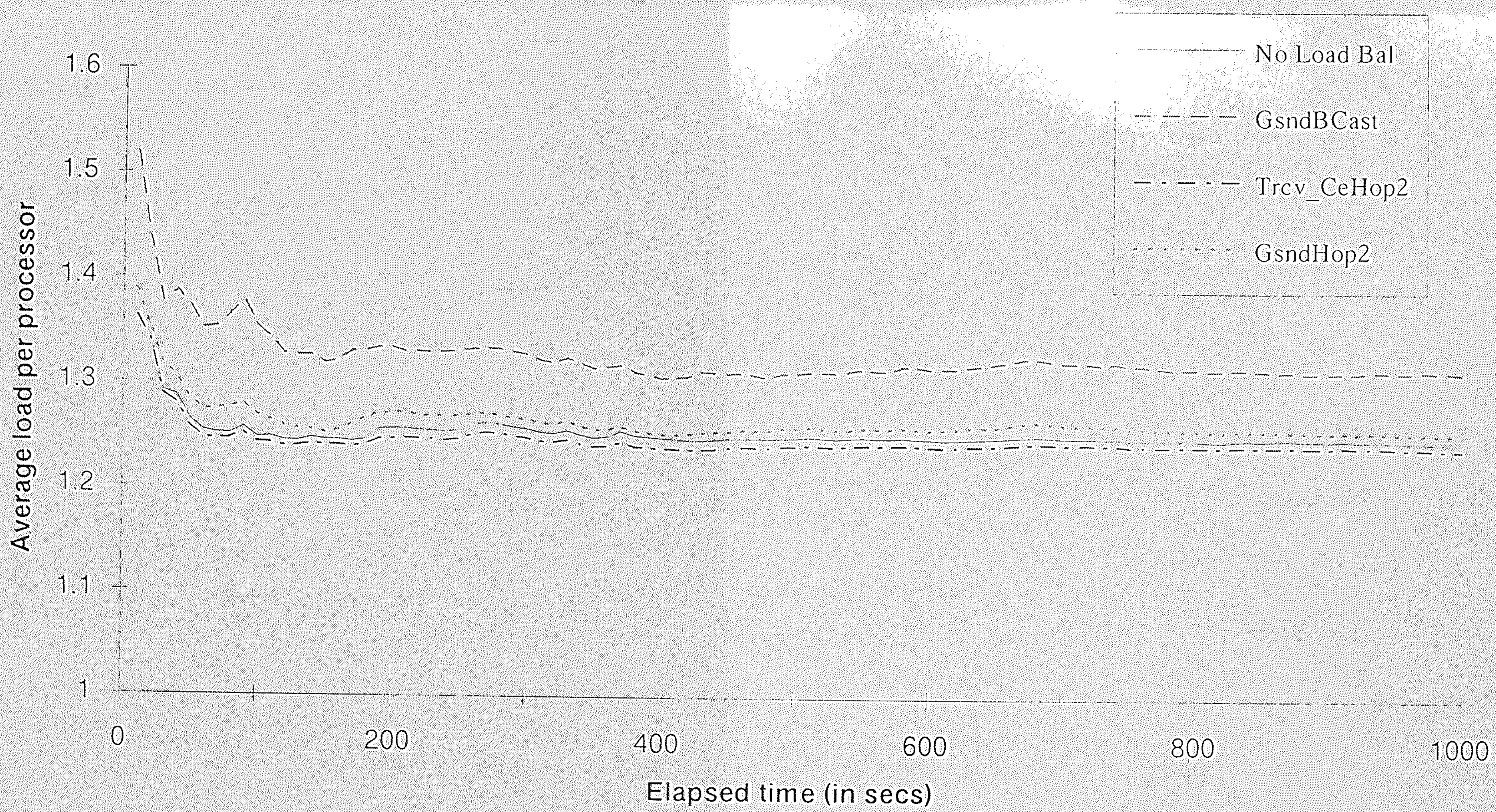


Figure B.1.11 16-Processor Model : Workload Convergence Profile ($\varrho = 0.2$)

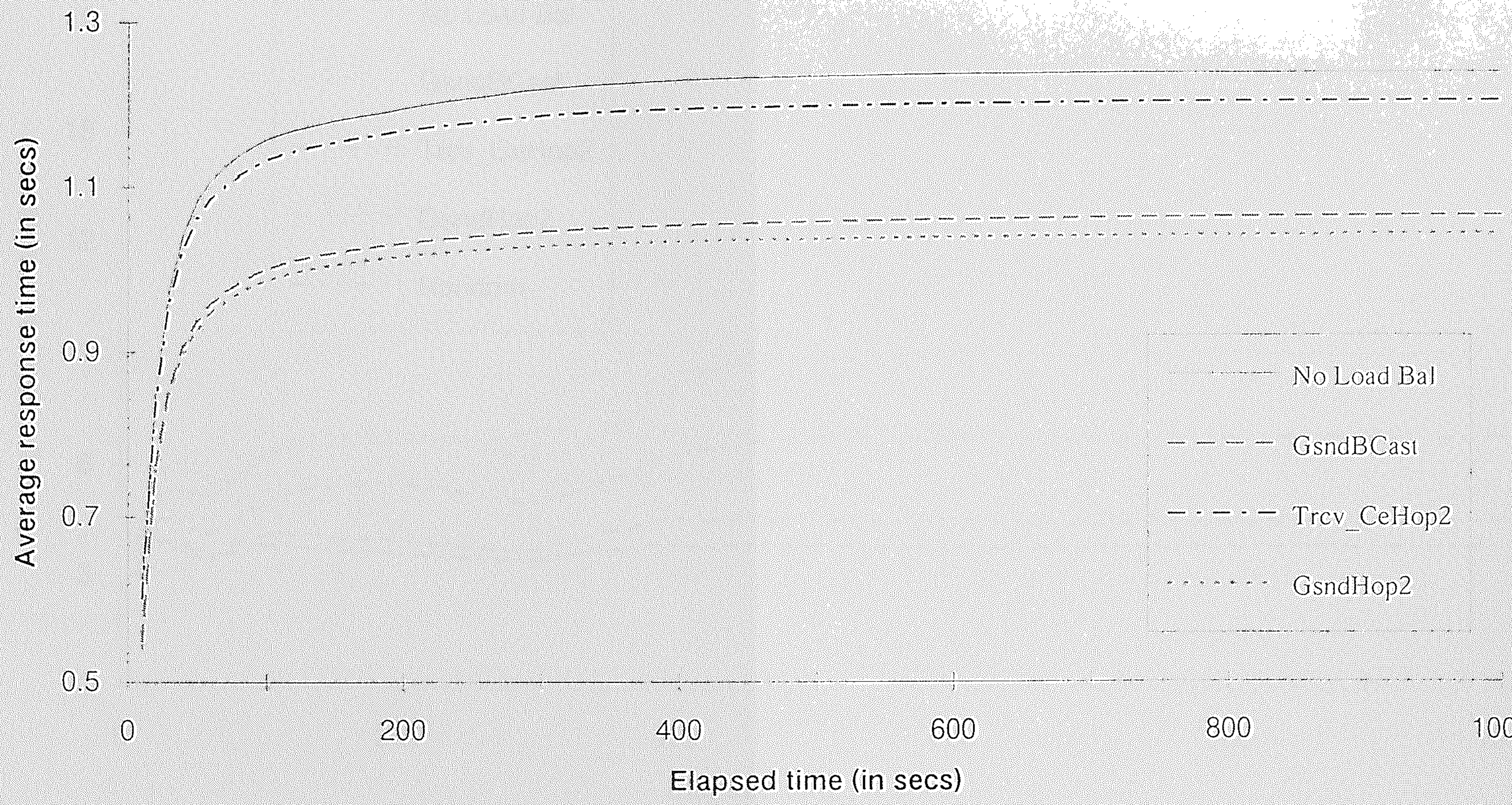


Figure B.1.12 16-Processor Model : Response Time Convergence Profile ($\varrho = 0.2$)

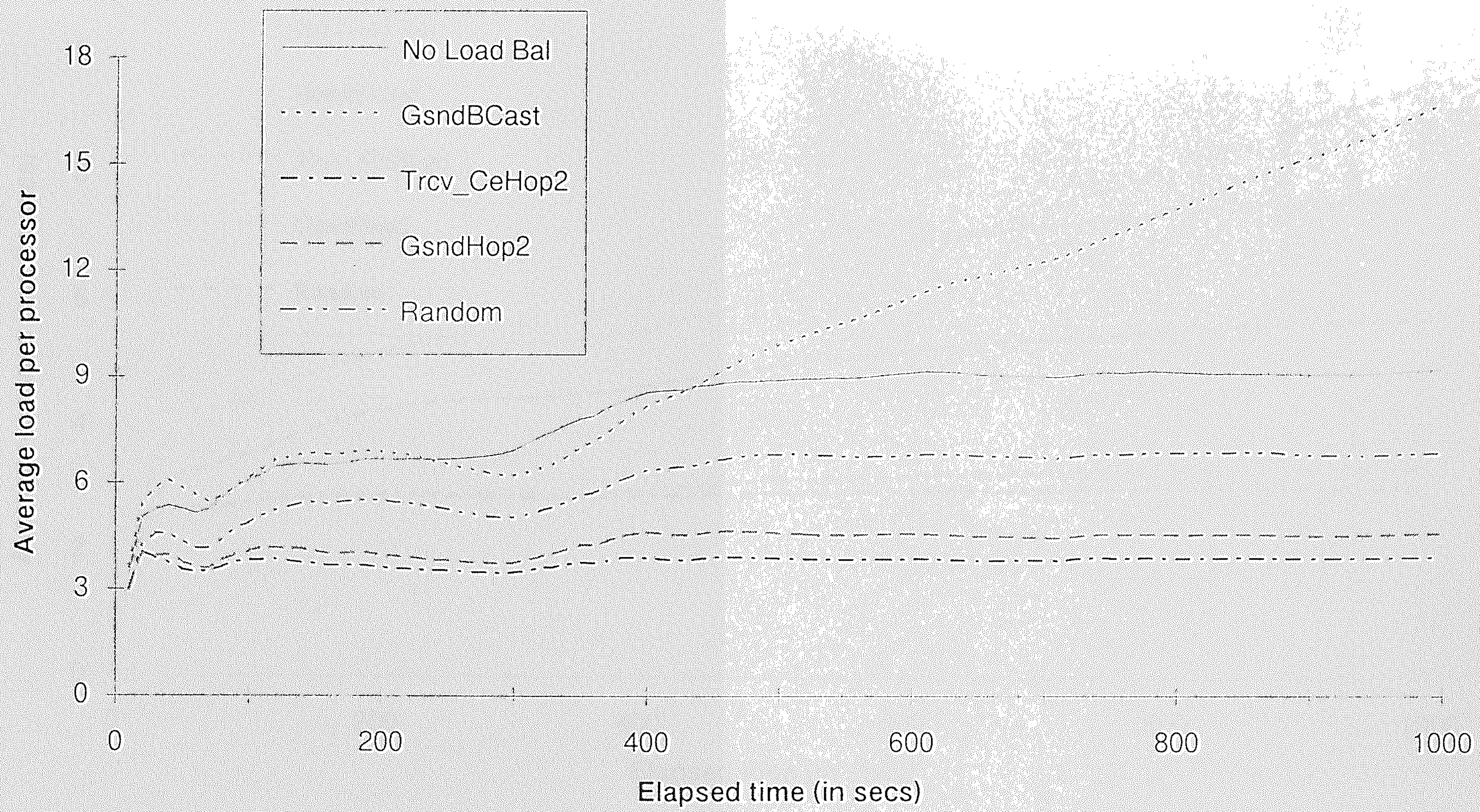


Figure B.1. 13 16-Processor Model : Workload Convergence Profile ($\varrho = 0.9$)

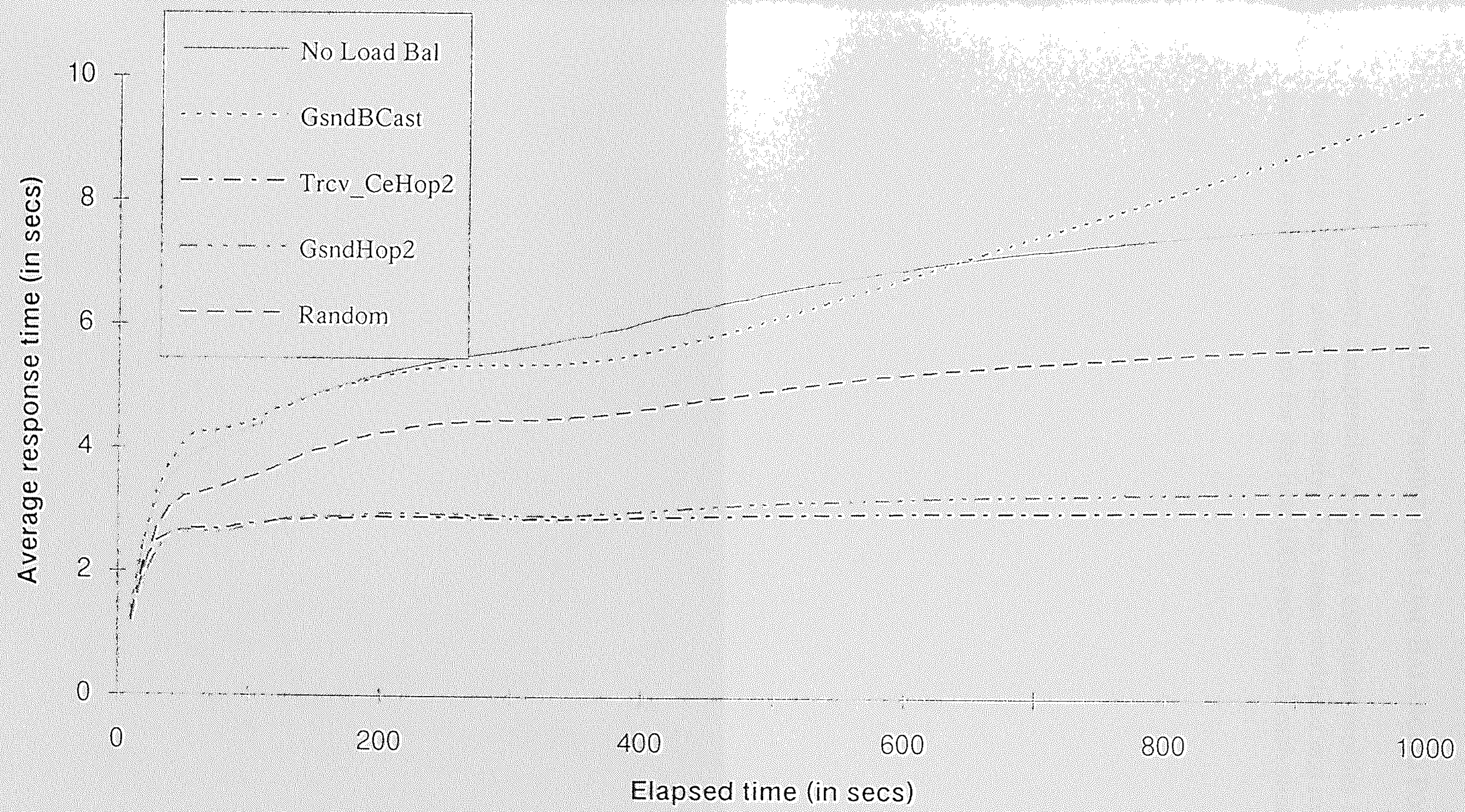


Figure B.1.14 16-Processor Model : Response Time Convergence Profile ($\varrho = 0.9$)

B.2 Graphical Execution Profile For Load Balancing Policies

This section presents comparative graphical execution profiles for a selection of load balancing policies used in the study. The mean workload and run time performance is sampled every 10 seconds for the first 1000 seconds of each simulation run.

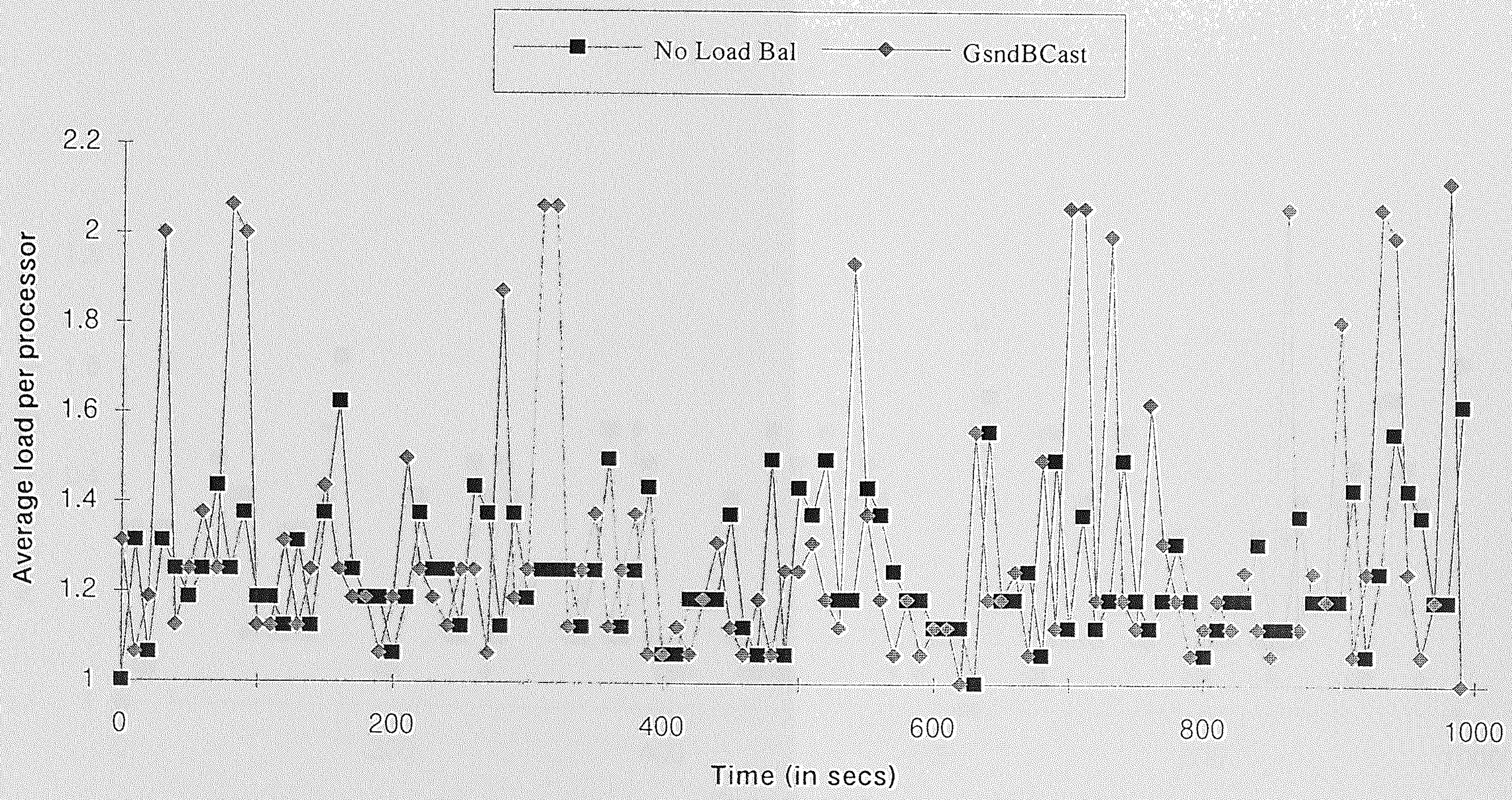


Figure B.2.1 No Load Bal vs Global Broadcast : Lightweight Process Workload Profile ($\varrho = 0.2$)

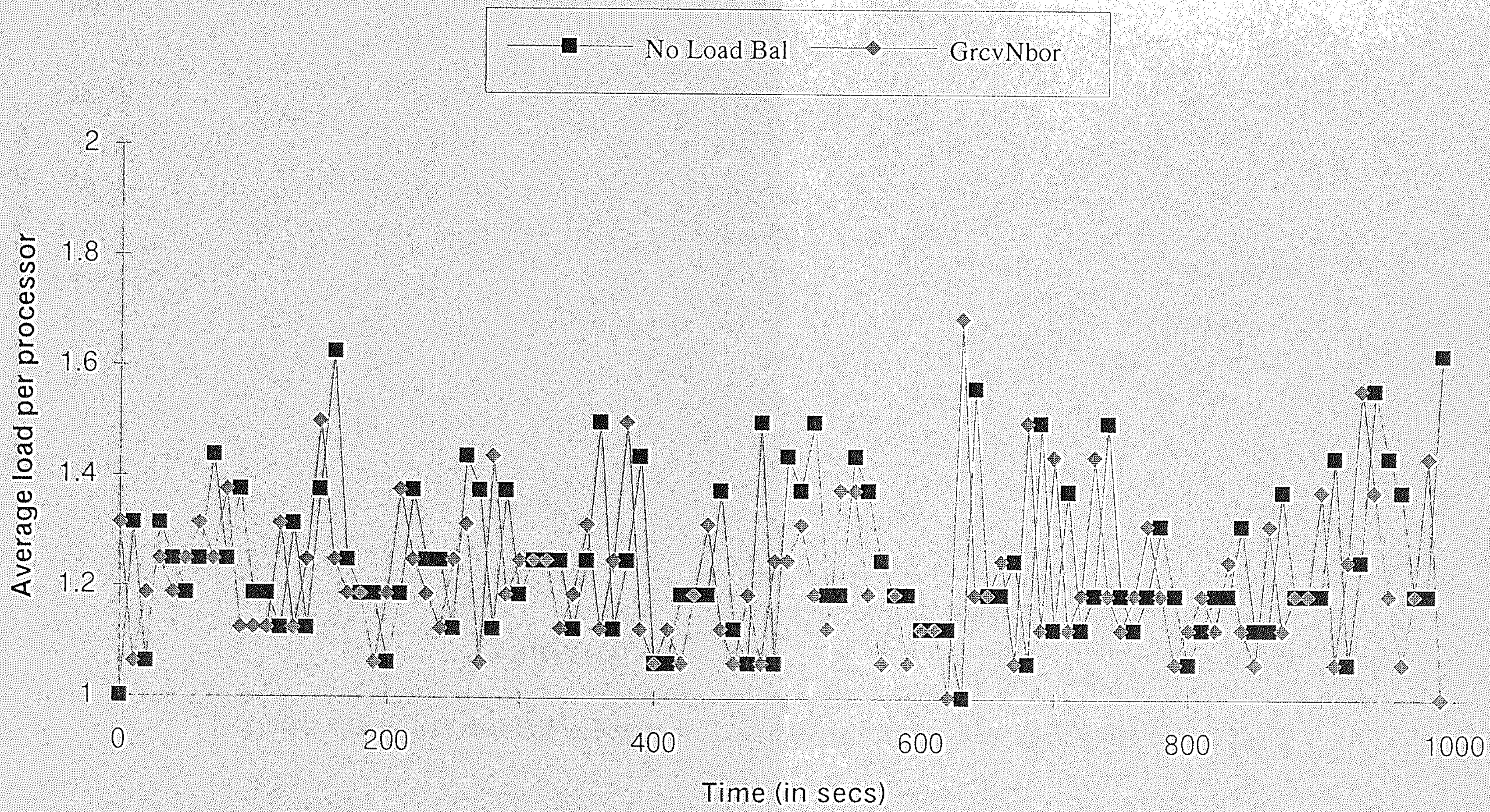


Figure B.2.2 No Load Bal vs Global (rcv) Neighbour : Lightweight Process Workload Profile (ϱ = 0.2)

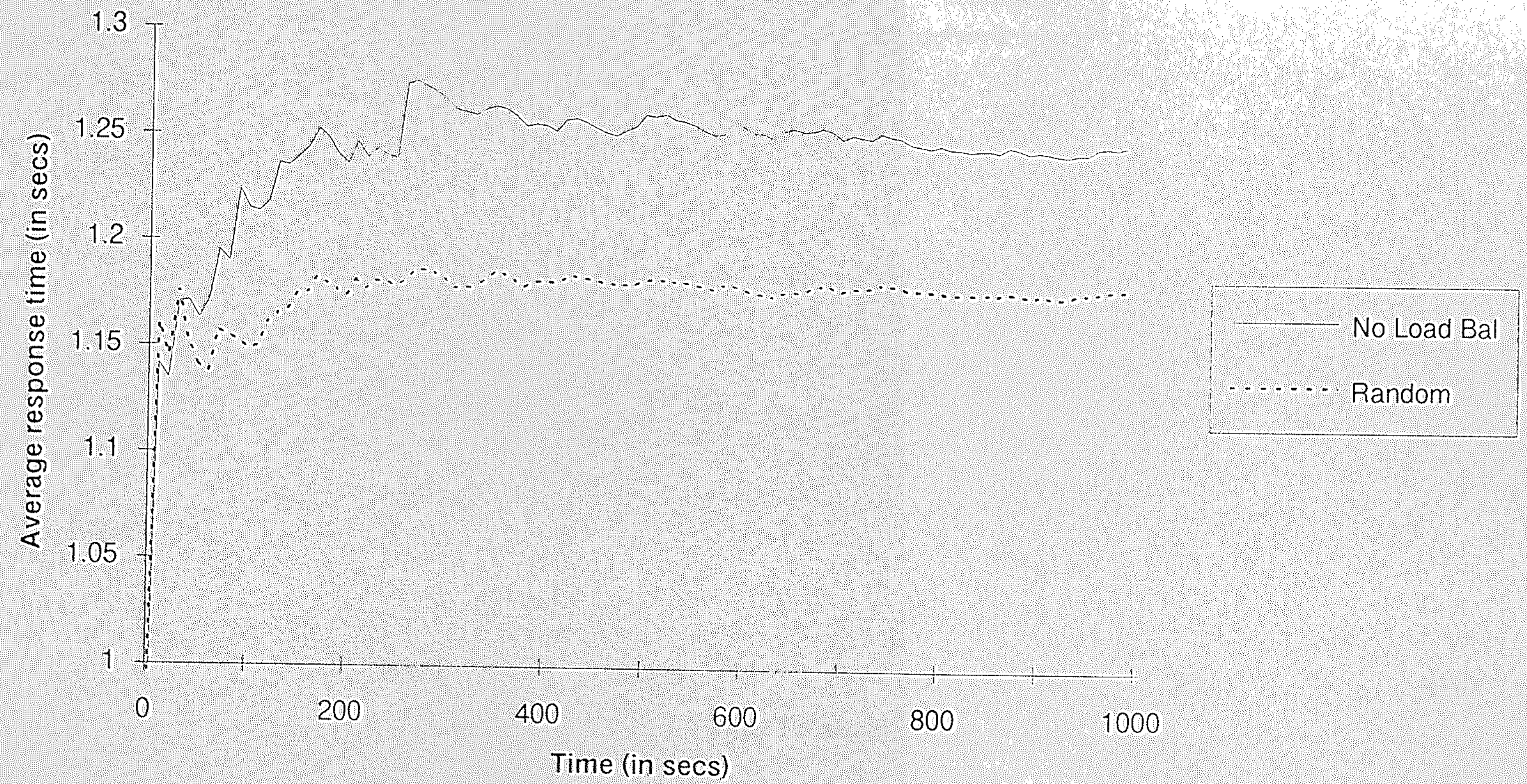


Figure B.2.3 No Load Bal vs Random : Lightweight Process Runtime Profile ($\varrho = 0.2$)

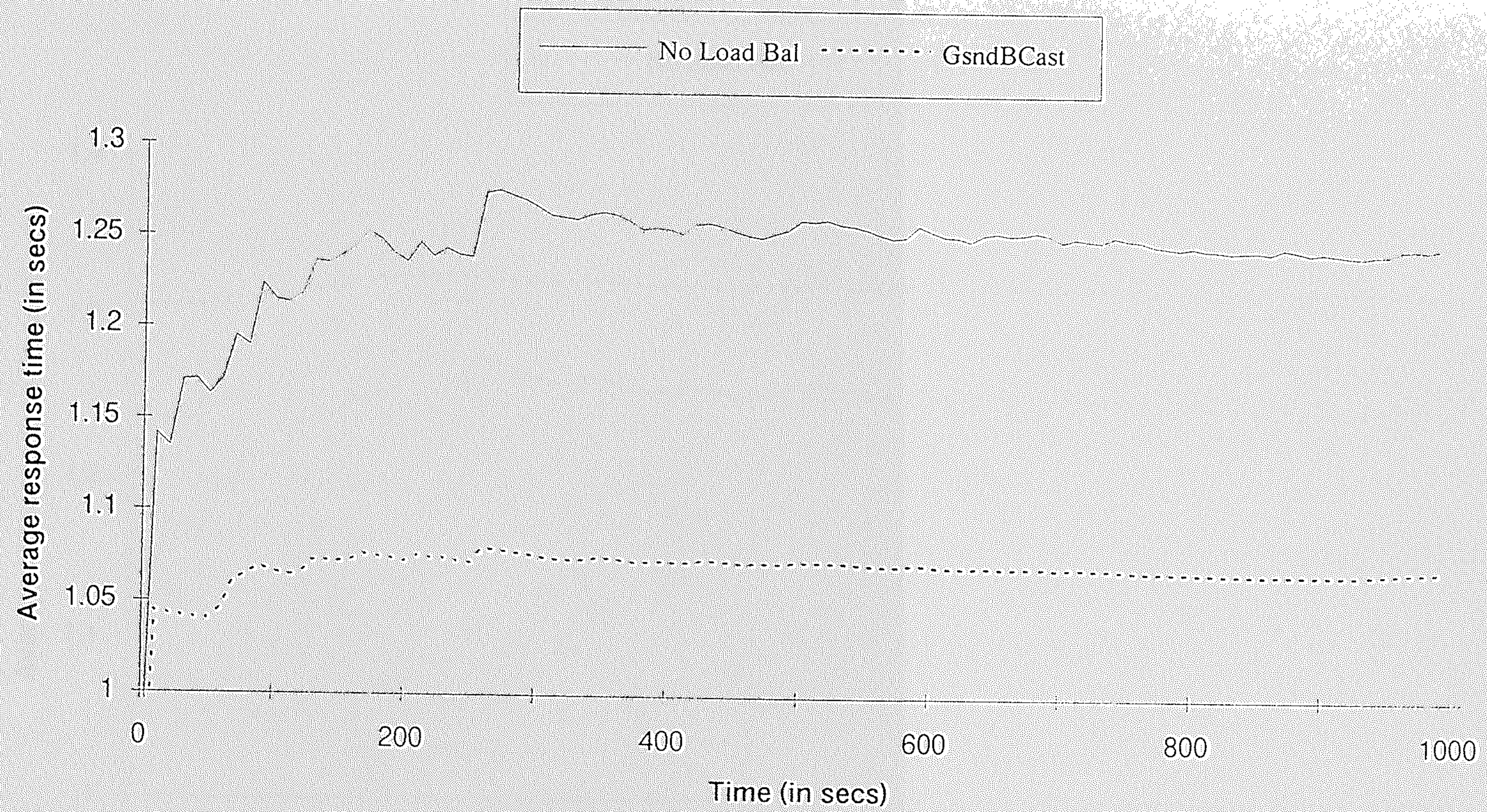


Figure B.2.4 No Load Bal vs Global (snd) Broadcast : Lightweight Process Runtime Profile ($\varrho = 0.2$)

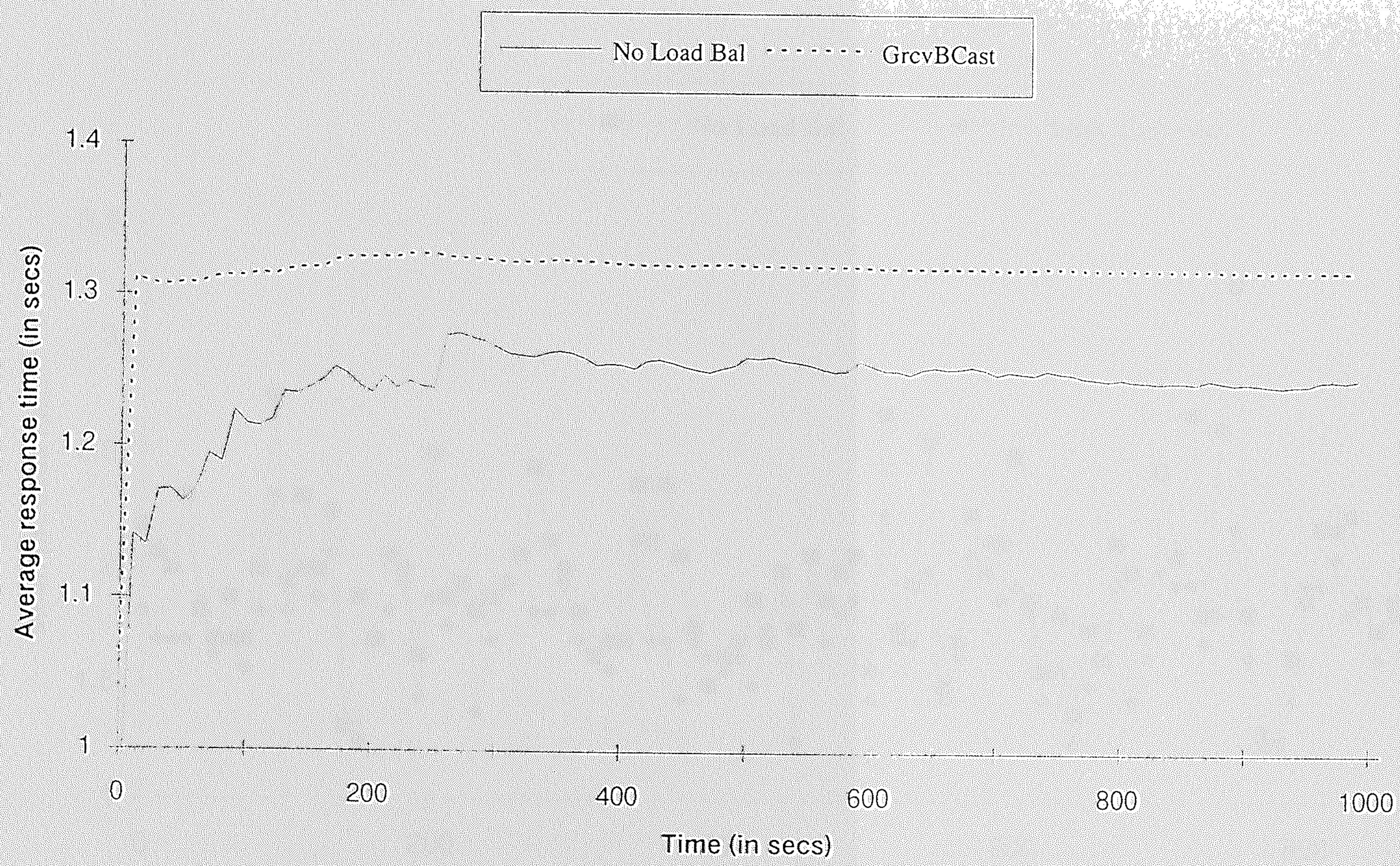


Figure B.2.5 No Load Bal vs Global (rcv) Broadcast : Lightweight Process Runtime Profile ($\varrho = 0.2$)

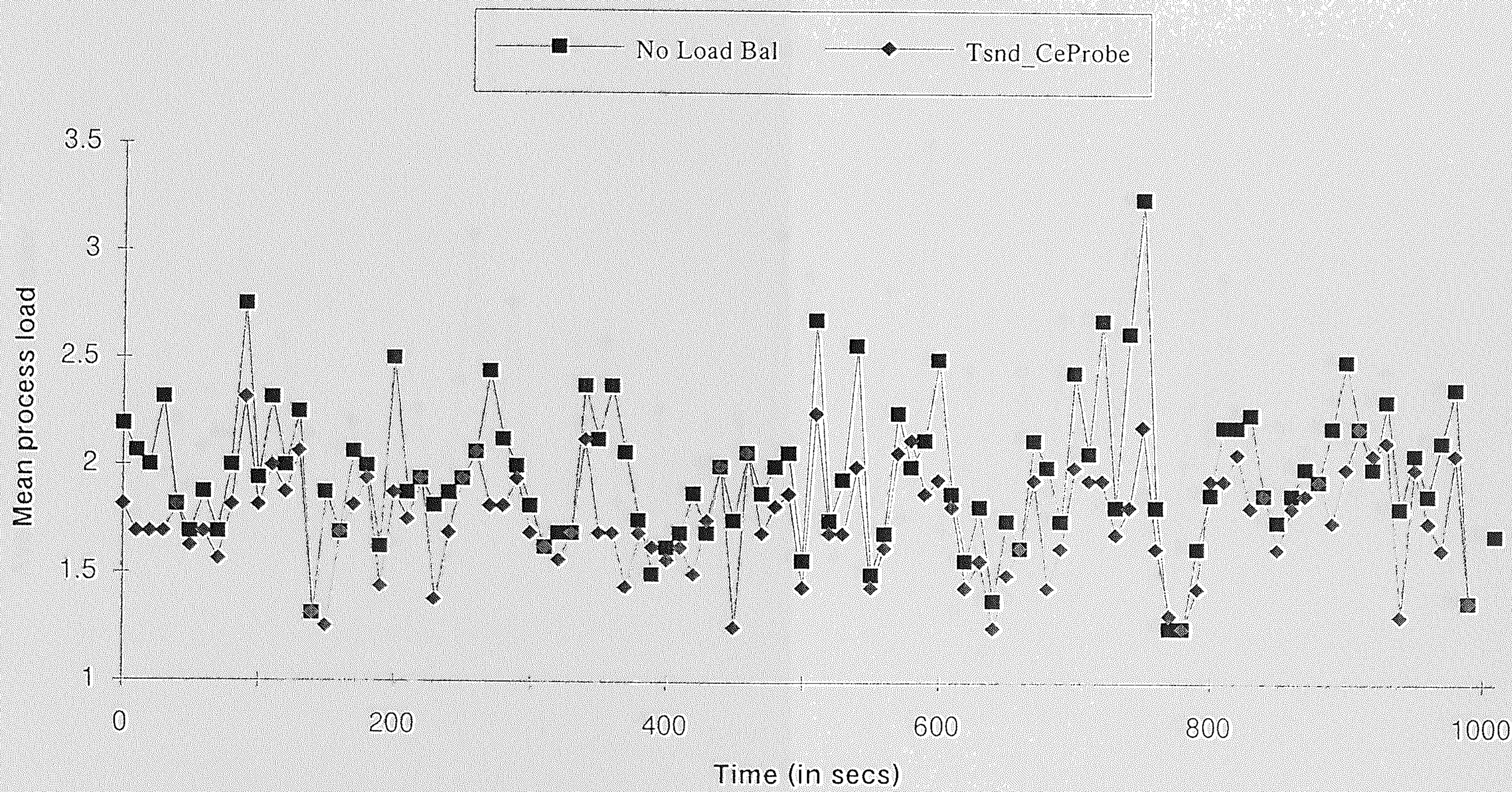


Figure B.2.6 No Load Bal vs Threshold (Cset) : Lightweight Process Workload Profile ($\varrho = 0.5$)

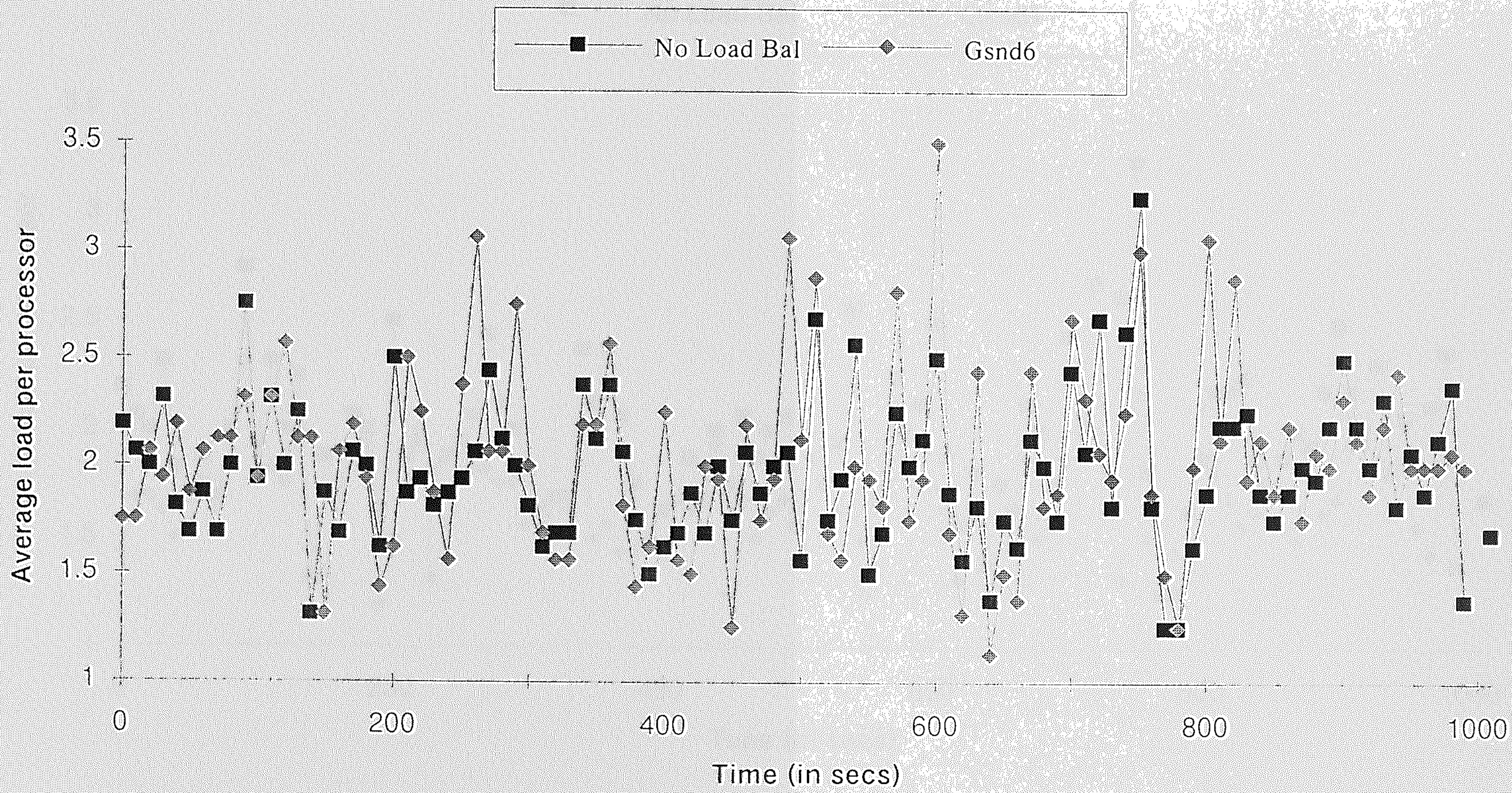


Figure B.2.7 No Load Bal vs Global (snd) Broadcast : Lightweight Process Workload Profile ($\varrho = 0.5$)

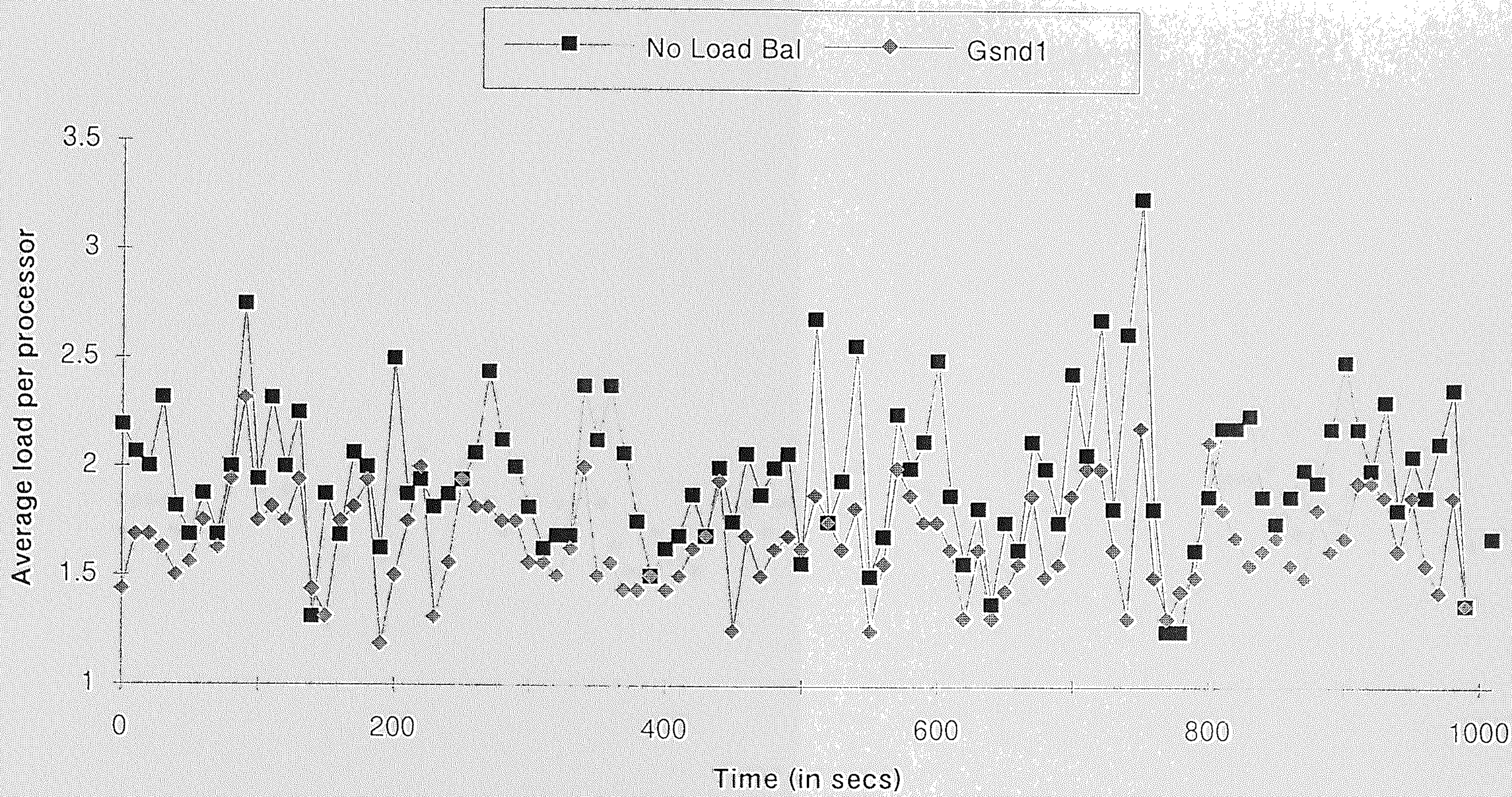


Figure B.2.8 No Load Bal vs Global (snd) Neighbour : Lightweight Process Workload Profile ($\rho = 0.5$)

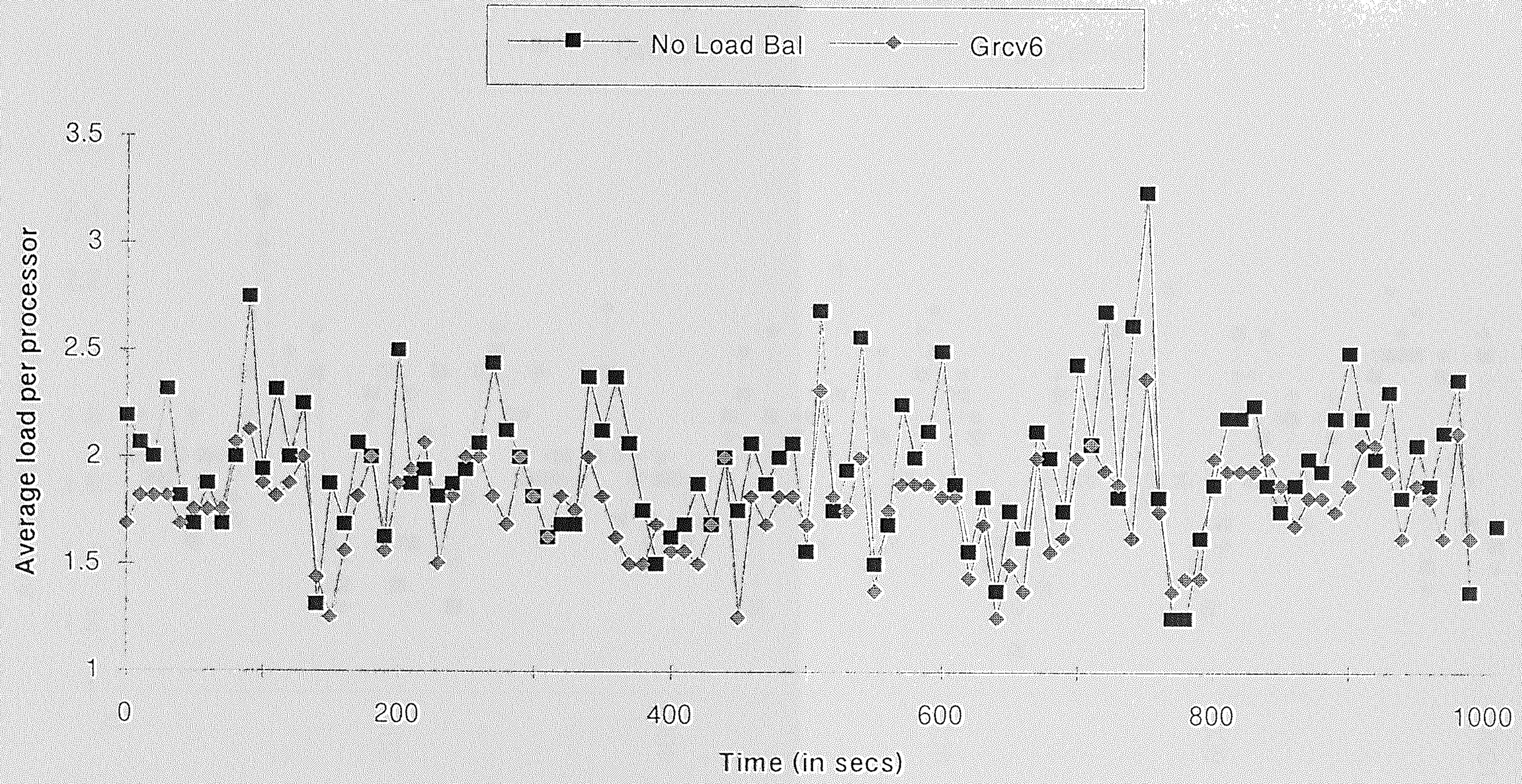


Figure B.2.9 No Load Bal vs Global (rcv) Broadcast : Lightweight Process Workload Profile ($\varrho = 0.5$)

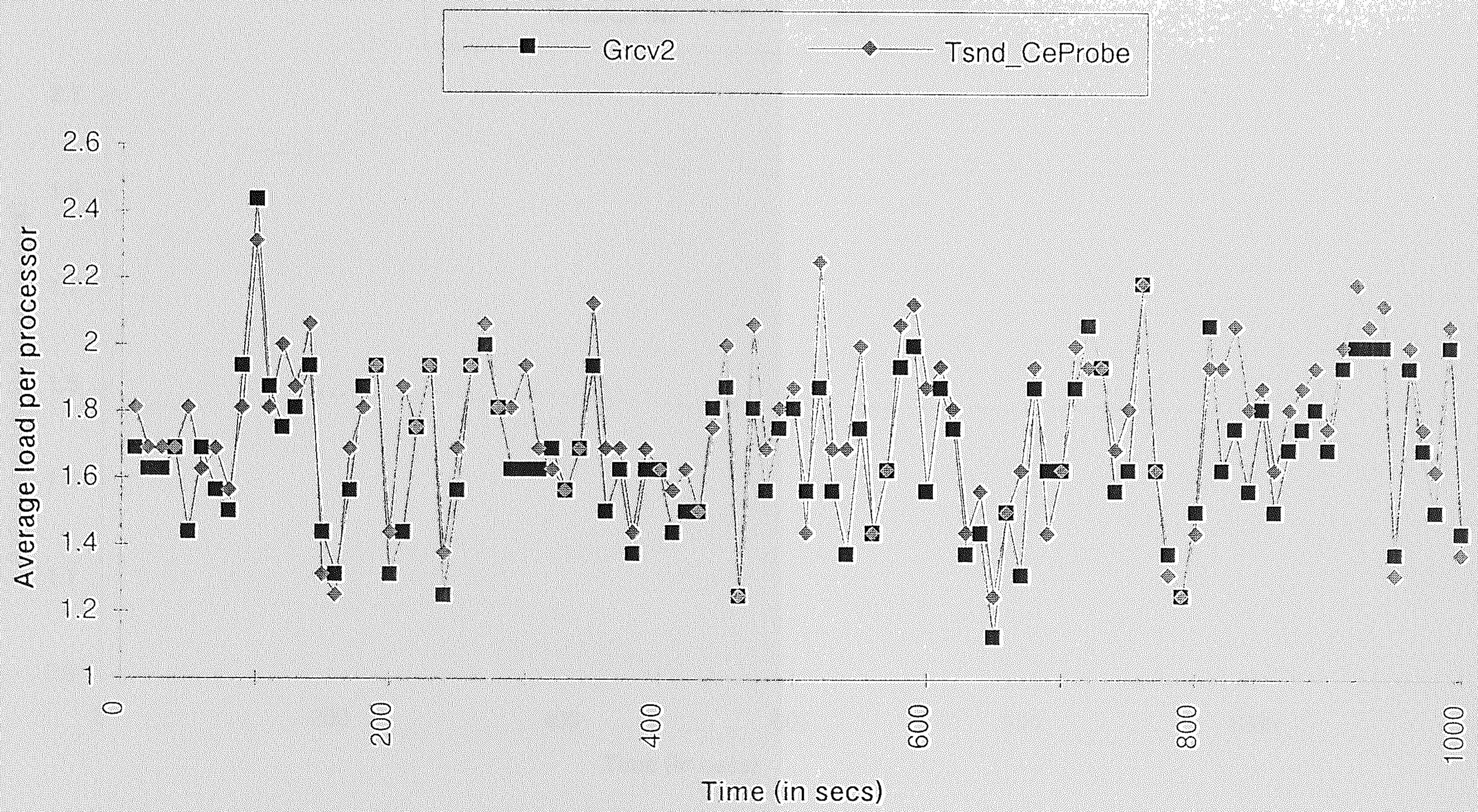


Figure B.2.10 Threshold (Cset) vs Global Receiver (Radius = 2 Hops) : Lightweight Process Workload Profile ($\varrho = 0.5$)

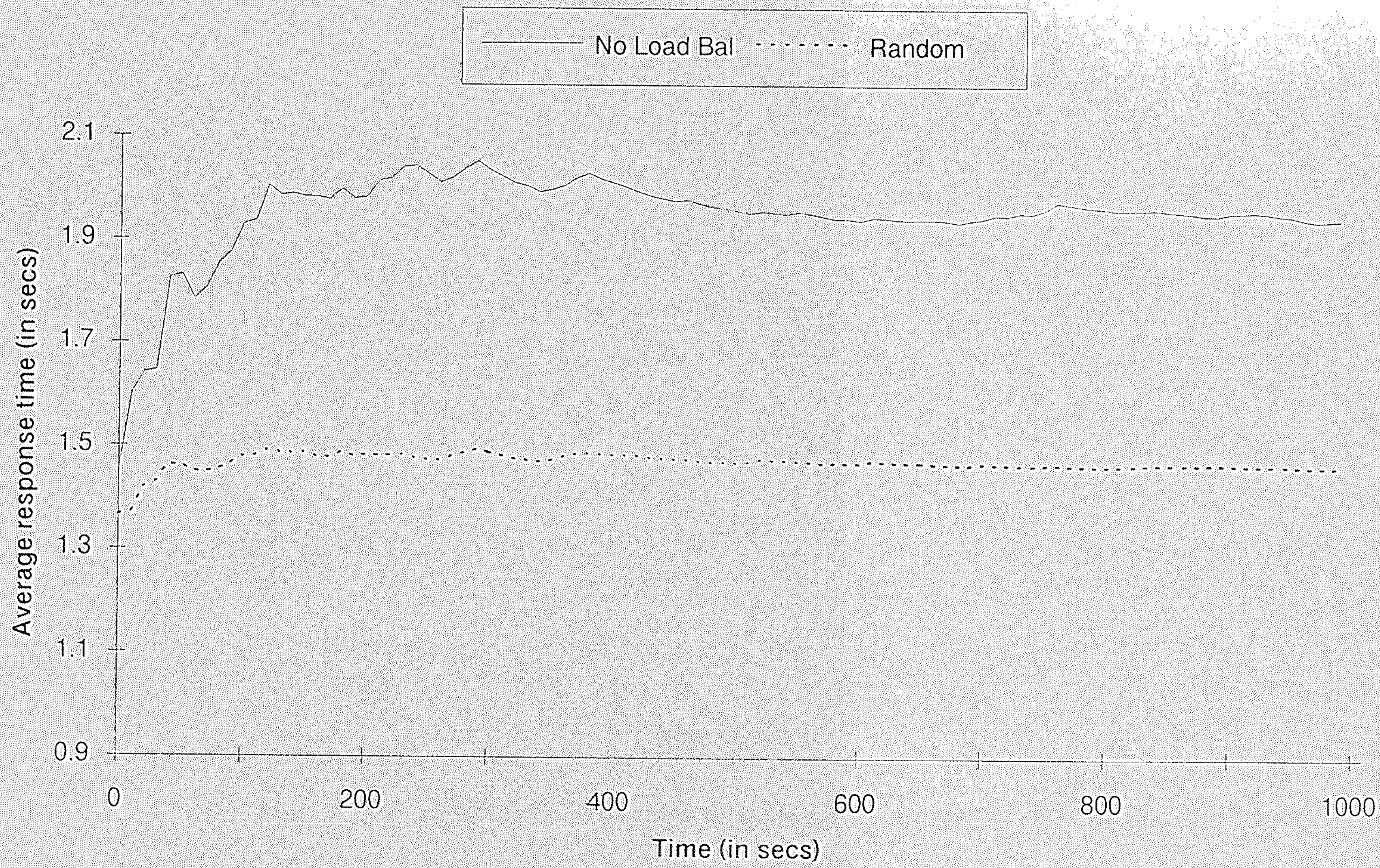


Figure B.2.11 No Load Bal vs Random : Lightweight Process Runtime Profile ($\varrho = 0.5$)

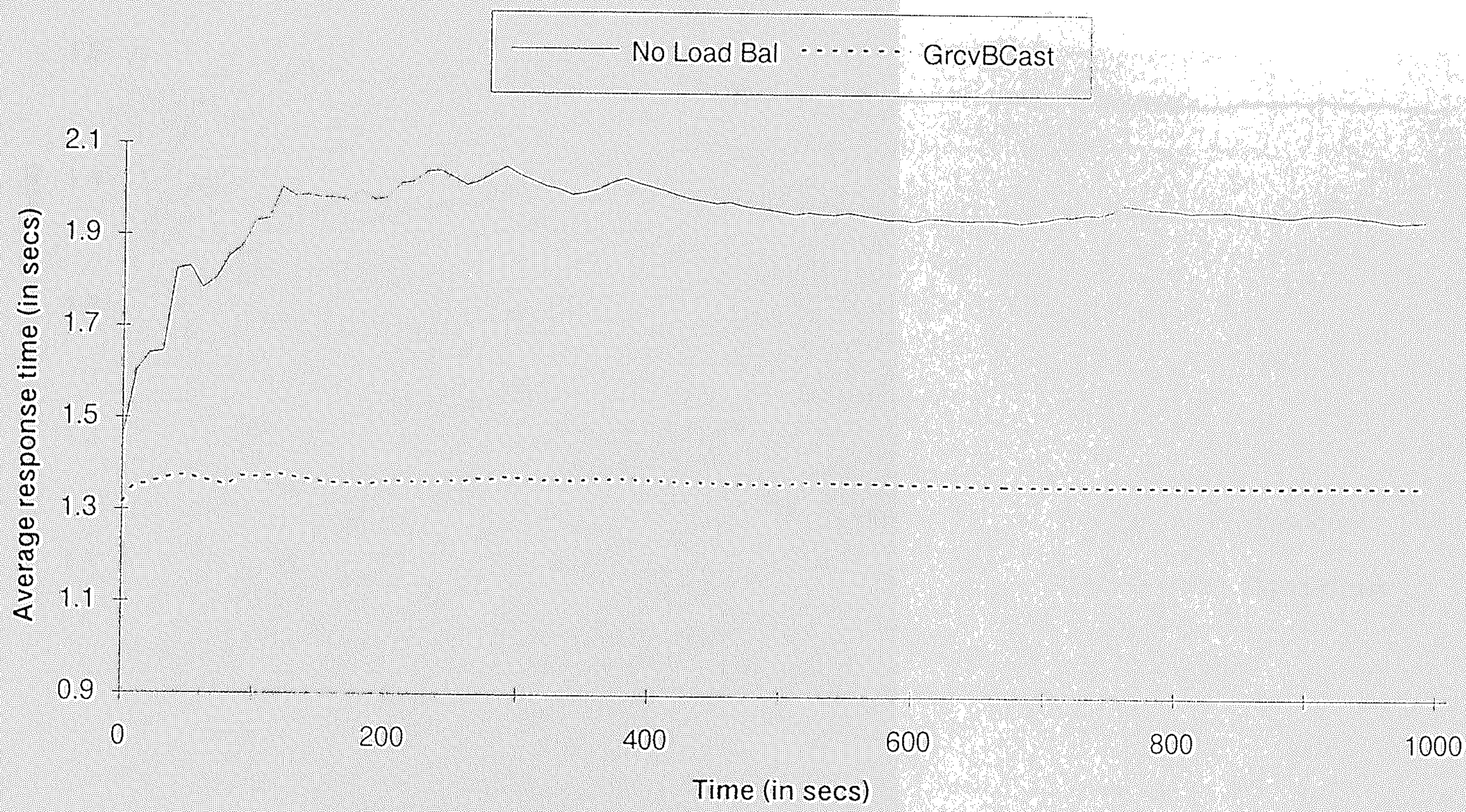


Figure B.2.12 No Load Bal vs Global (rcv) Broadcast : Lightweight Process Runtime Profile ($\varrho = 0.5$)

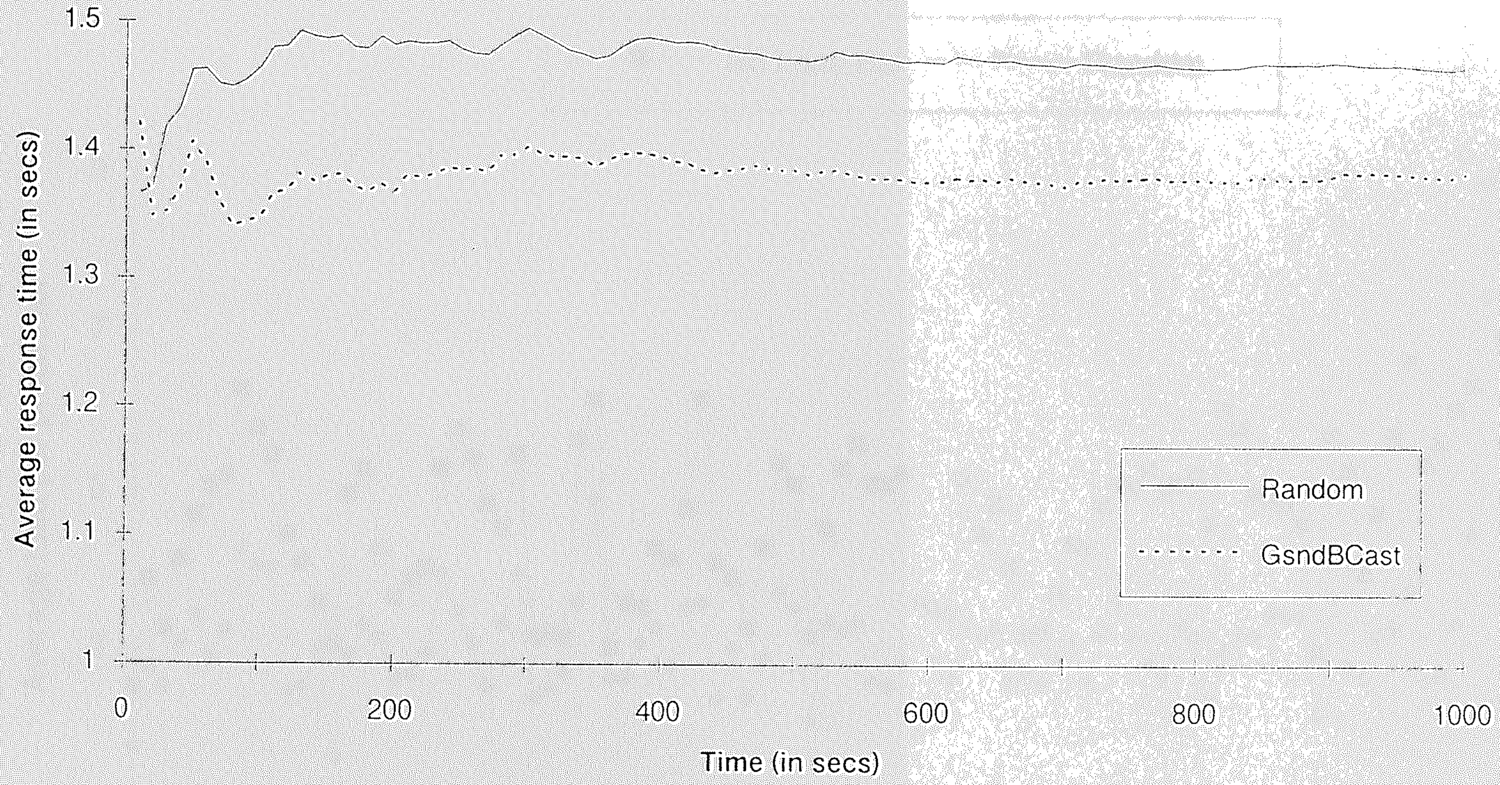


Figure B.2.13 Random vs Global (snd) Broadcast : Lightweight Process Runtime Profile ($\varrho = 0.5$)

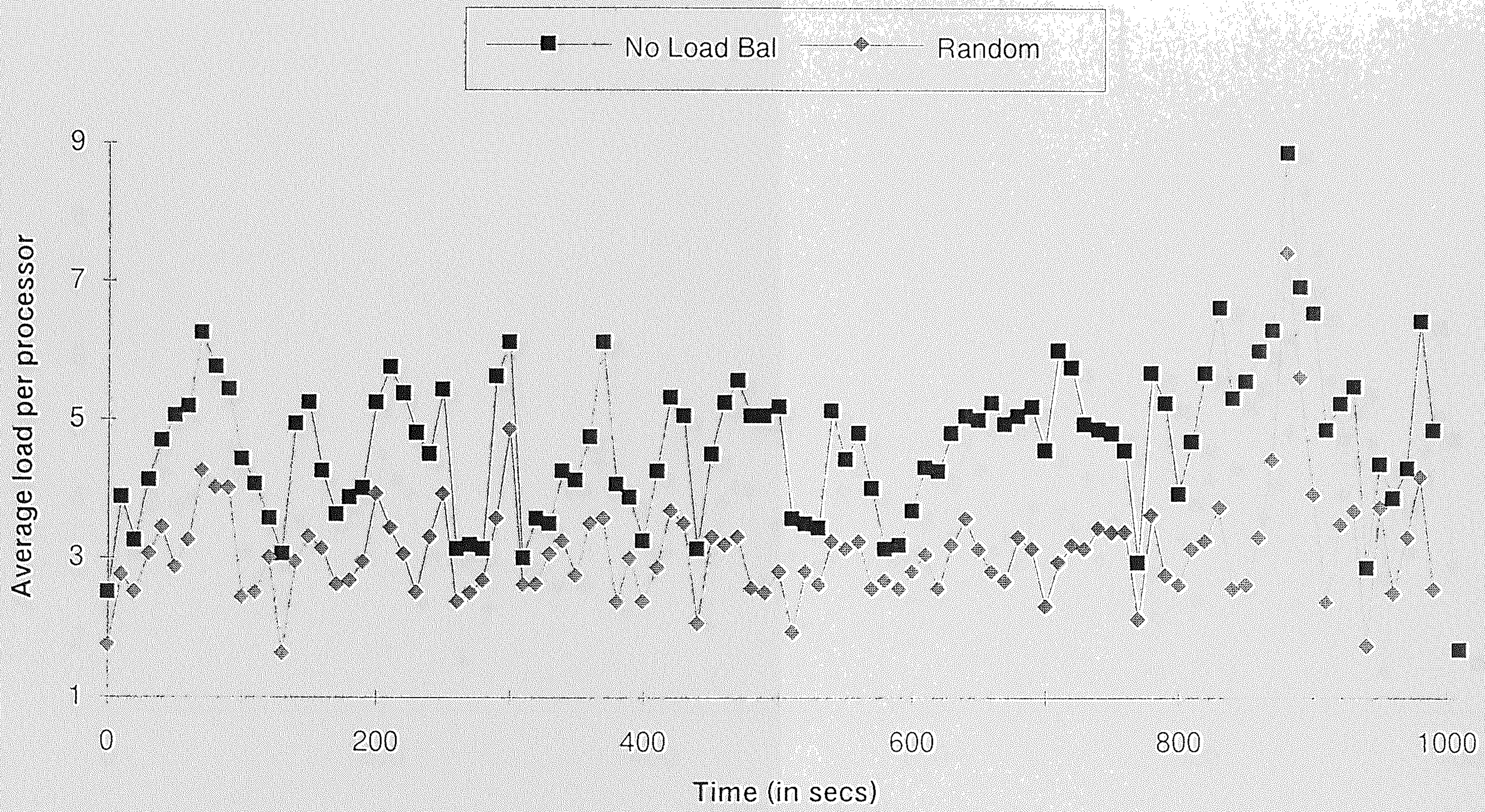


Figure B.2.14 No Load Bal vs Random : Lightweight Process Workload Profile ($\varrho = 0.8$)

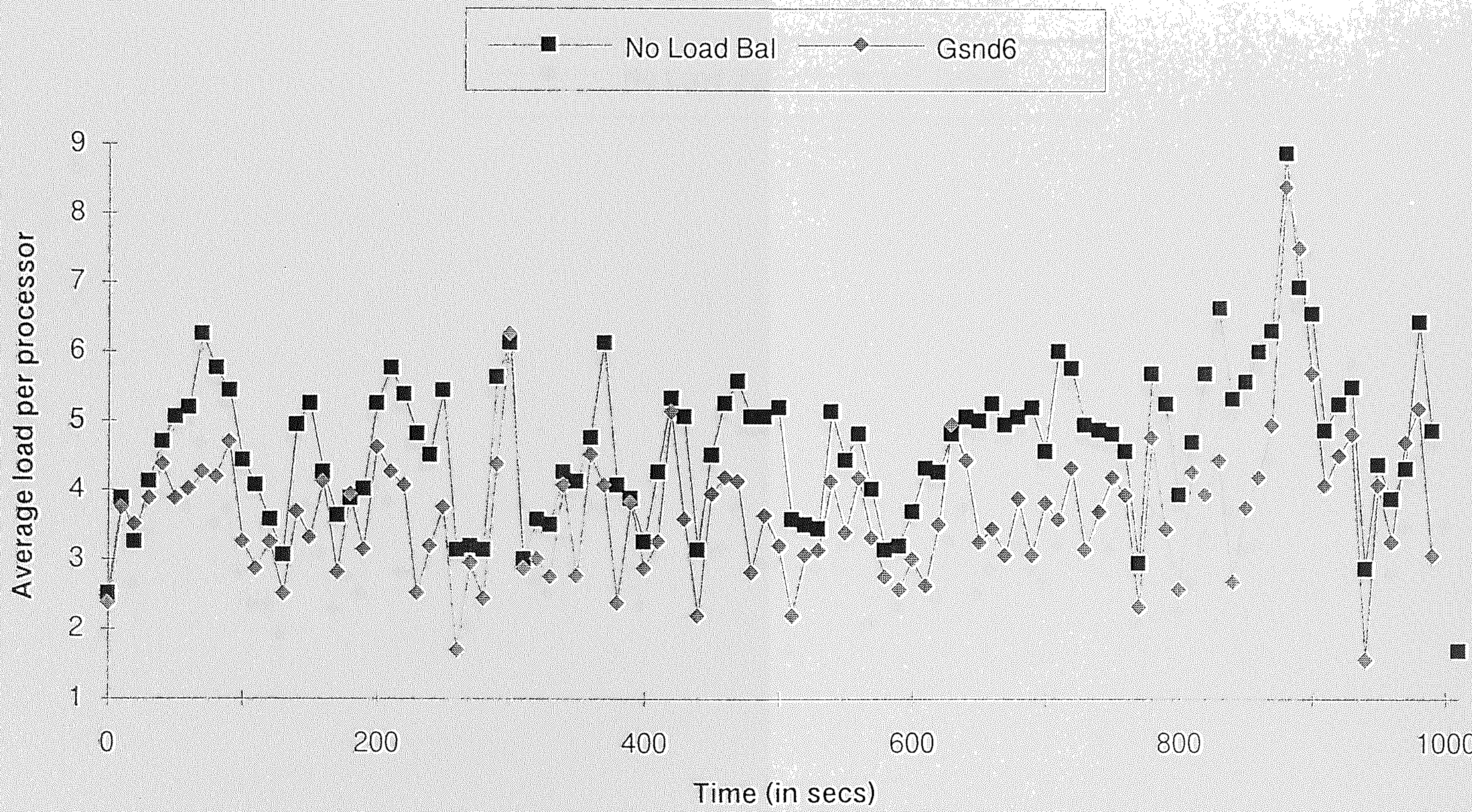


Figure B.2.15 No Load Bal vs Global (snd) Broadcast : Lightweight Process Workload Profile ($\varrho = 0.8$)

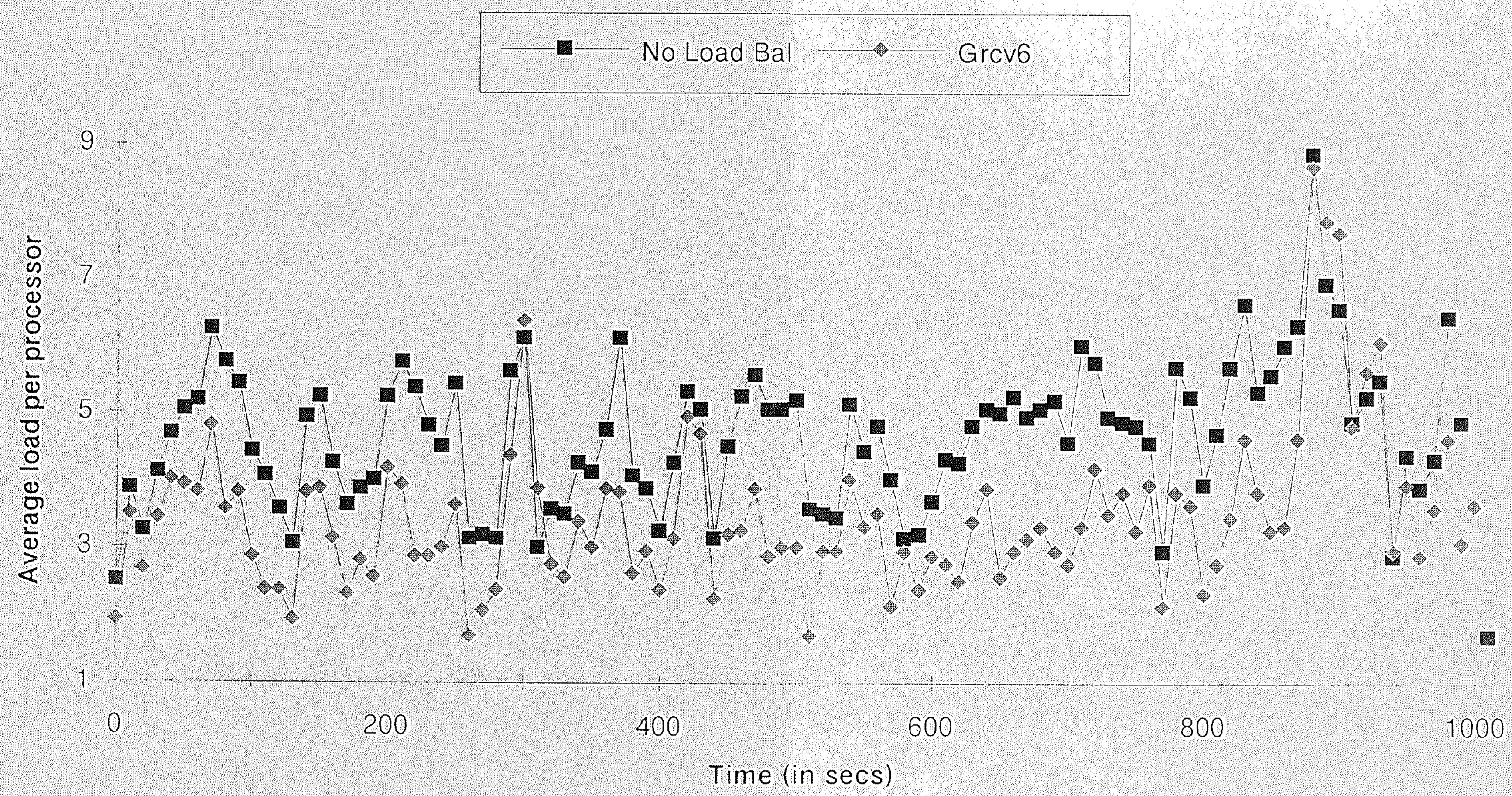


Figure B.2.16 No Load Bal vs Global (rcv) Broadcast : Lightweight Process Workload Profile ($\rho = 0.8$)

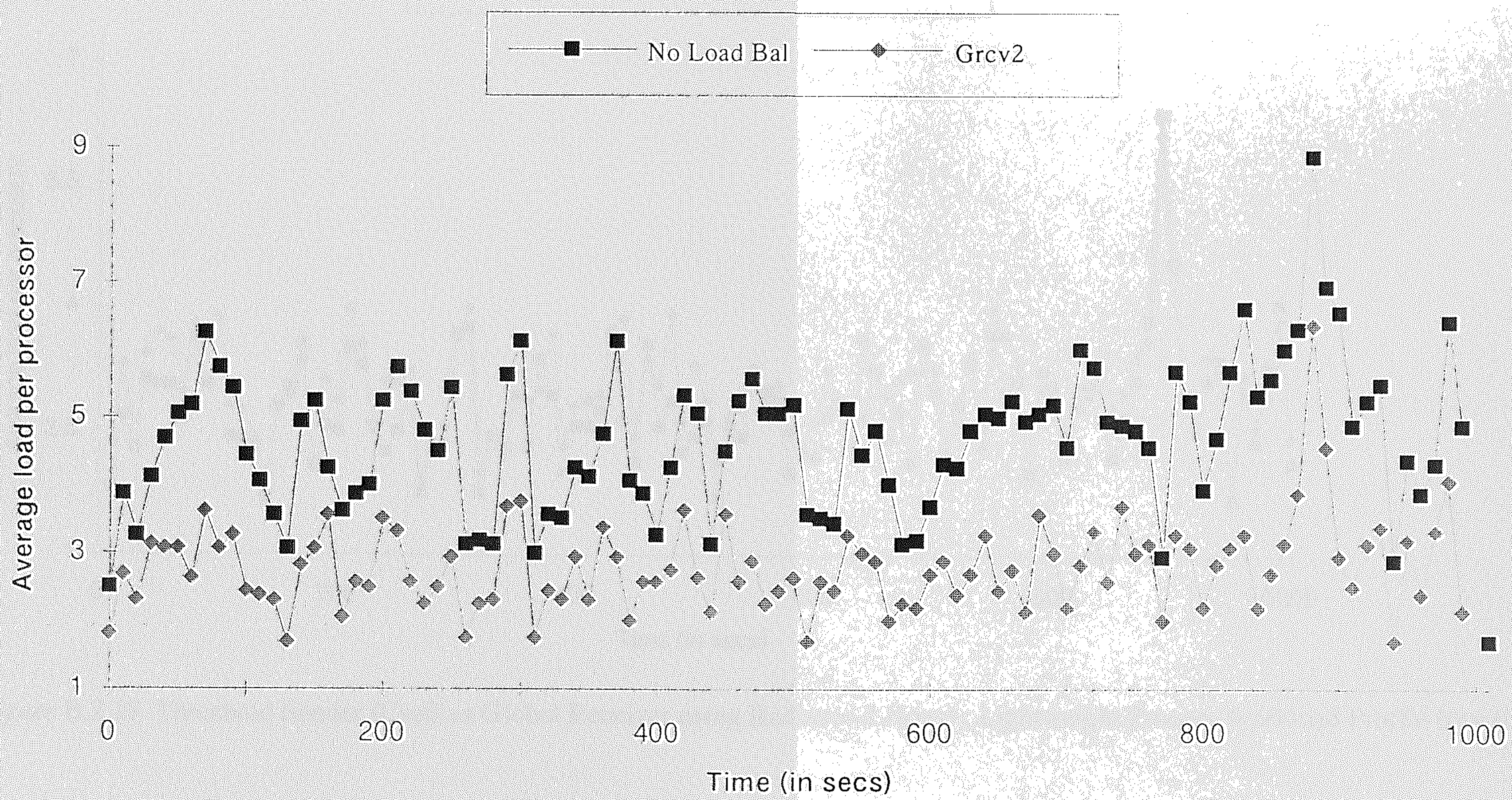


Figure B.2.17 No Load Bal vs Global Receiver (Radius = 2 Hops) : Lightweight Process Workload Profile (ϱ = 0.8)

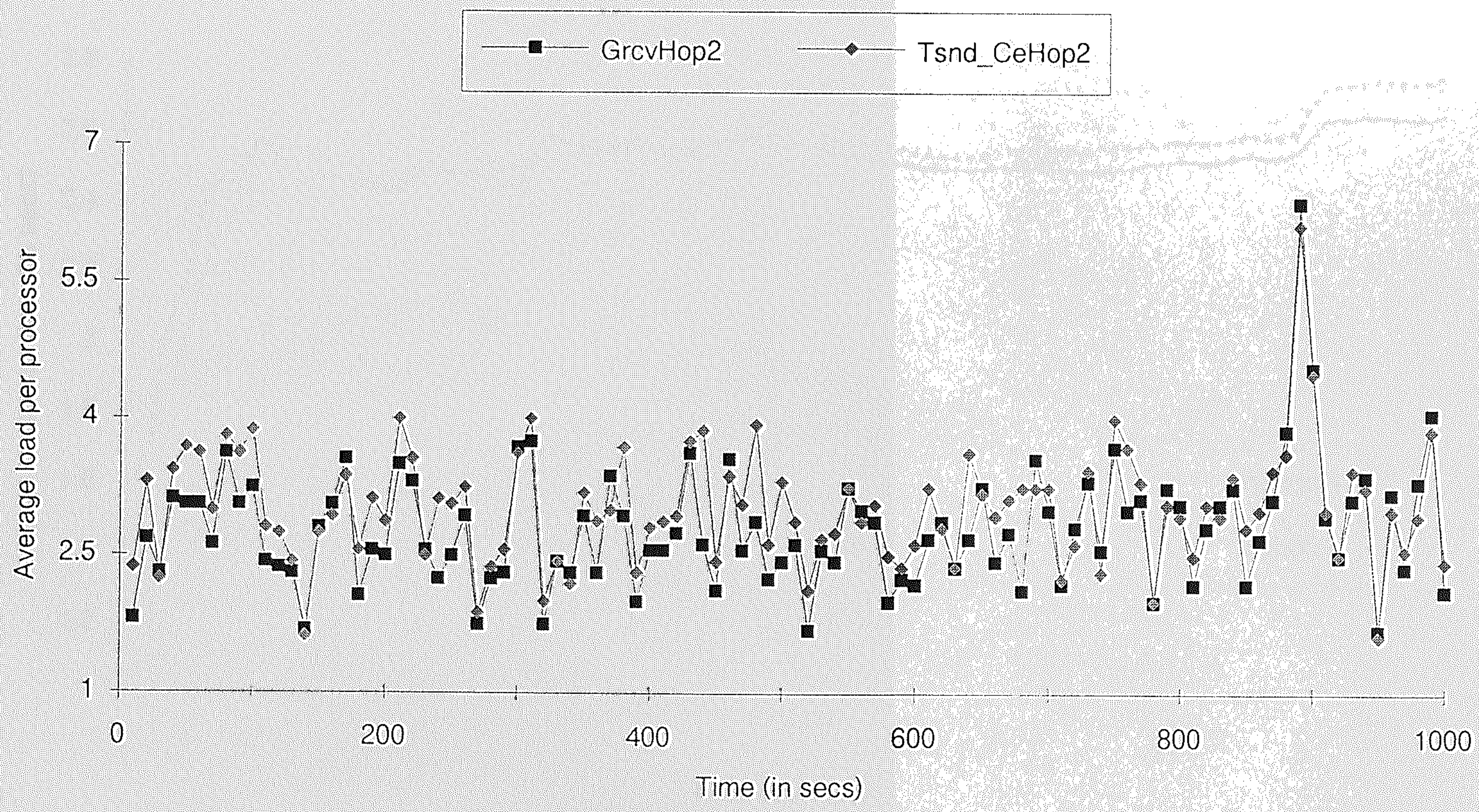


Figure B.2.18 Threshold Sender (Cset) vs Global Receiver using Radius = 2 Hops) : Lightweight Process Workload Profile (ρ = 0.8)

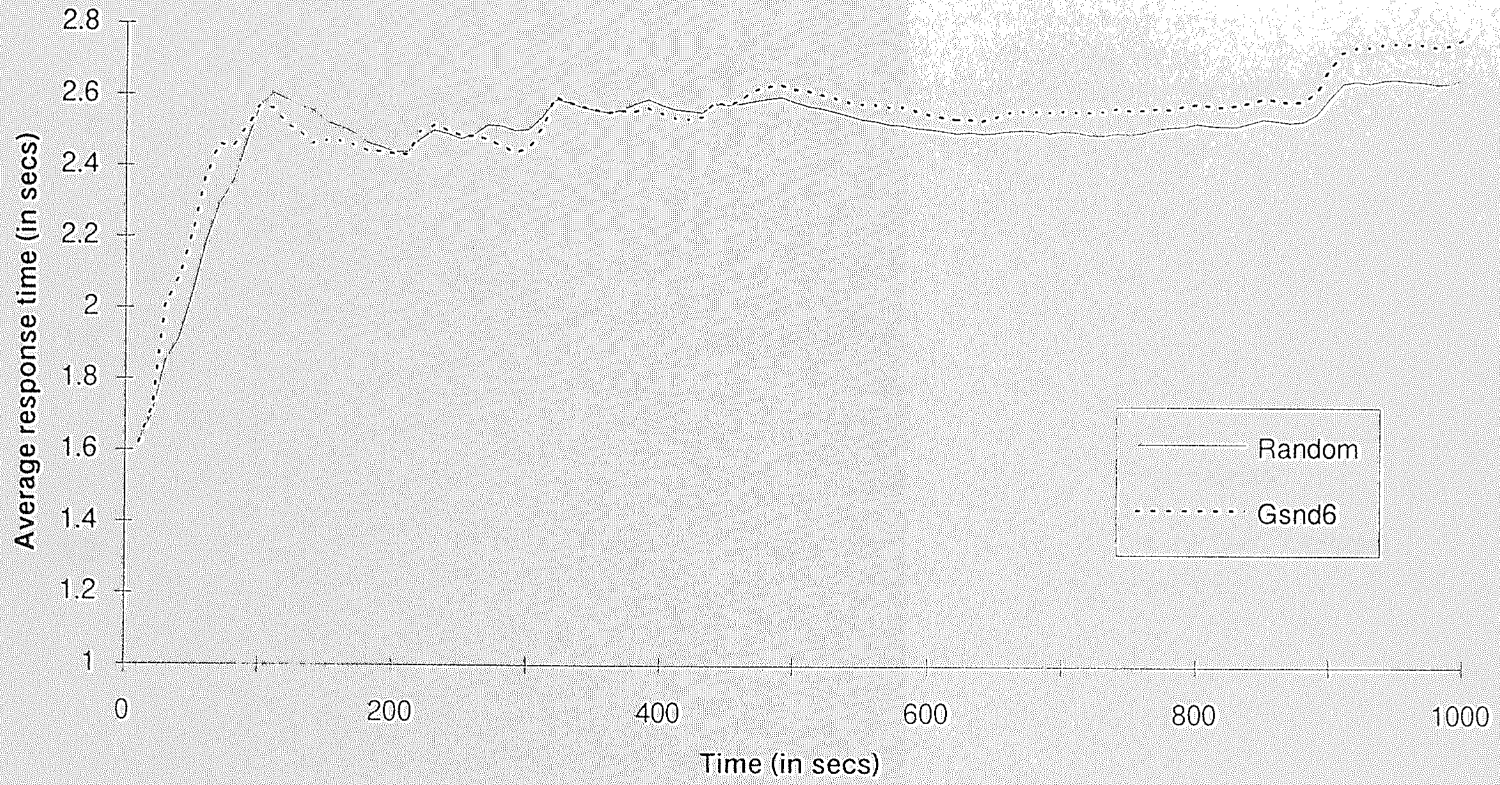


Figure B.2.19 Random vs Global (snd) Broadcast : Lightweight Process Runtime Profile ($\varrho = 0.8$)

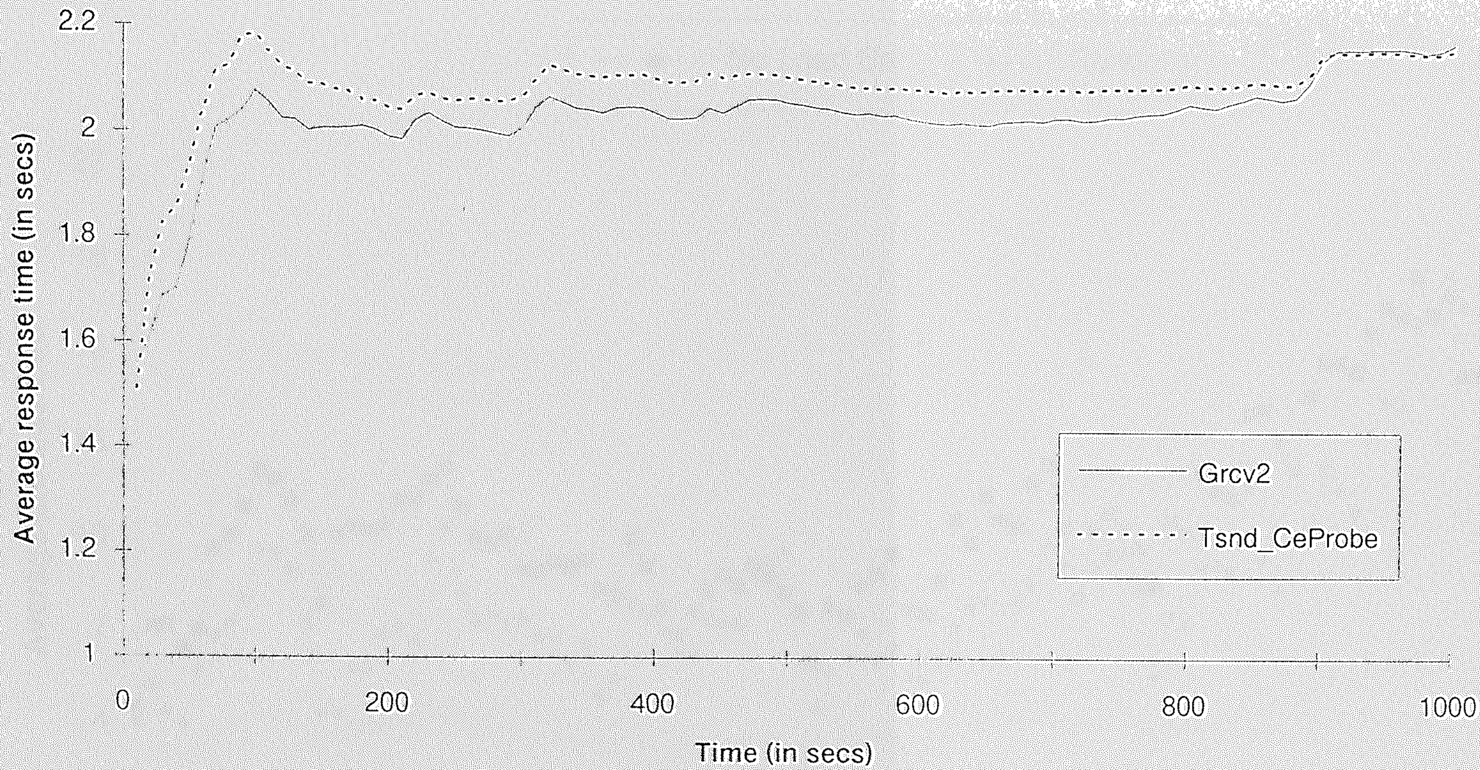


Figure B.2.20 Threshold (Cset) vs Global Receiver (Radius = 2 Hops) : Lightweight Process Runtime Profile ($\varrho = 0.8$)

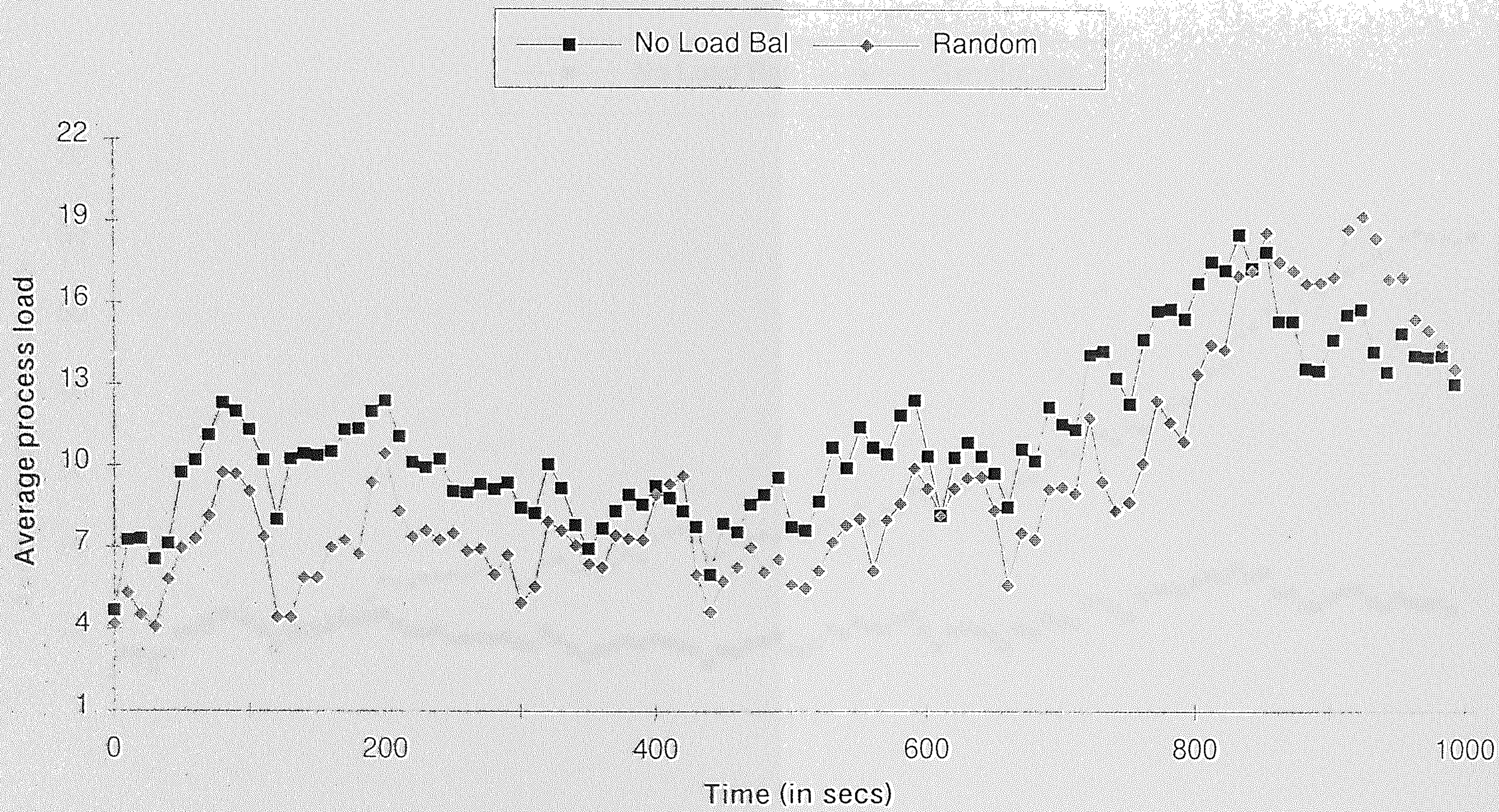


Figure B.2.21 No Load Bal vs Random : Lightweight Process Workload Profile ($\varrho = 0.9$)

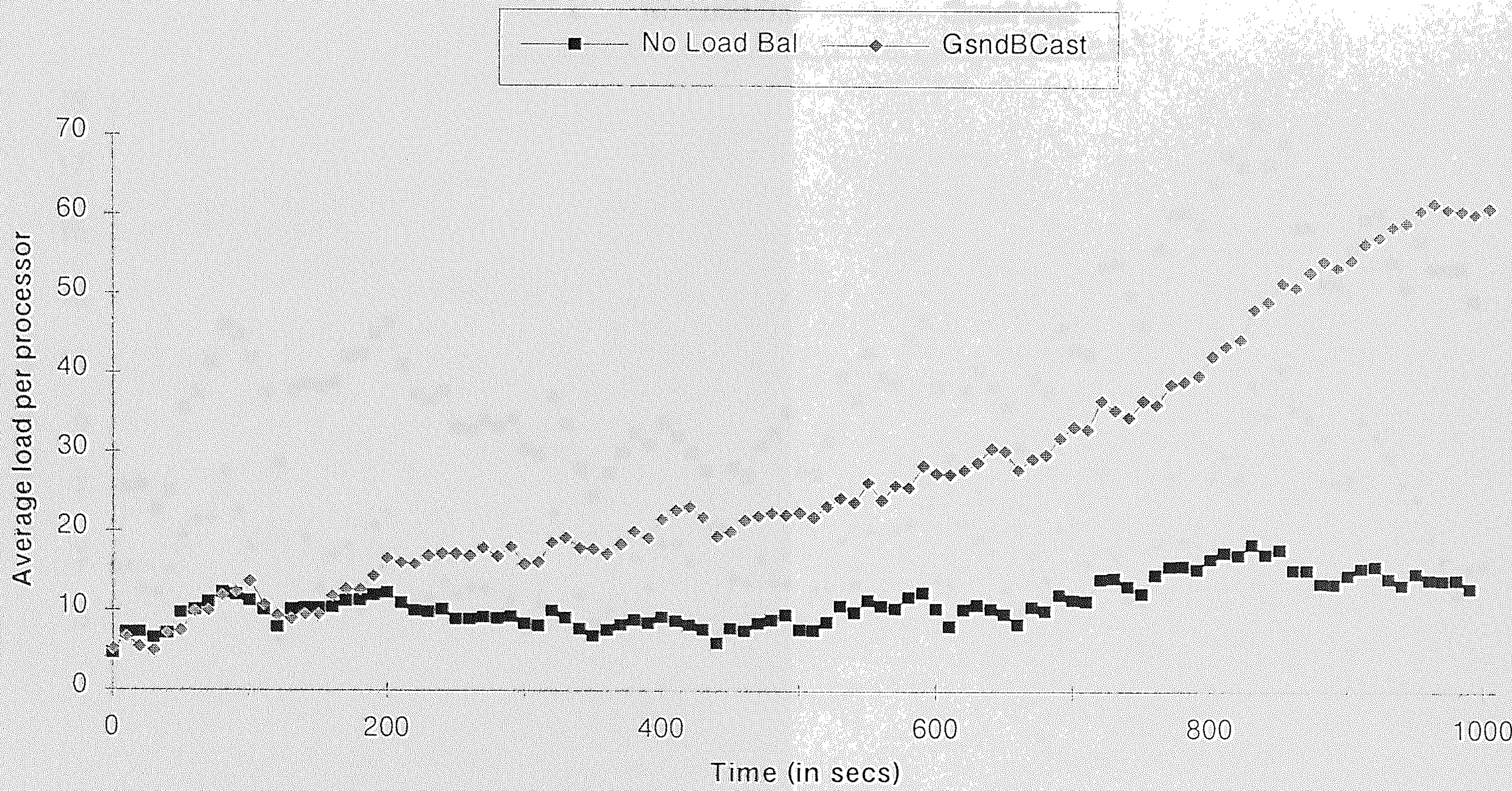


Figure B.2.22 No Load Bal vs Global (snd) Broadcast : Lightweight Process Workload Profile ($\varrho = 0.9$)

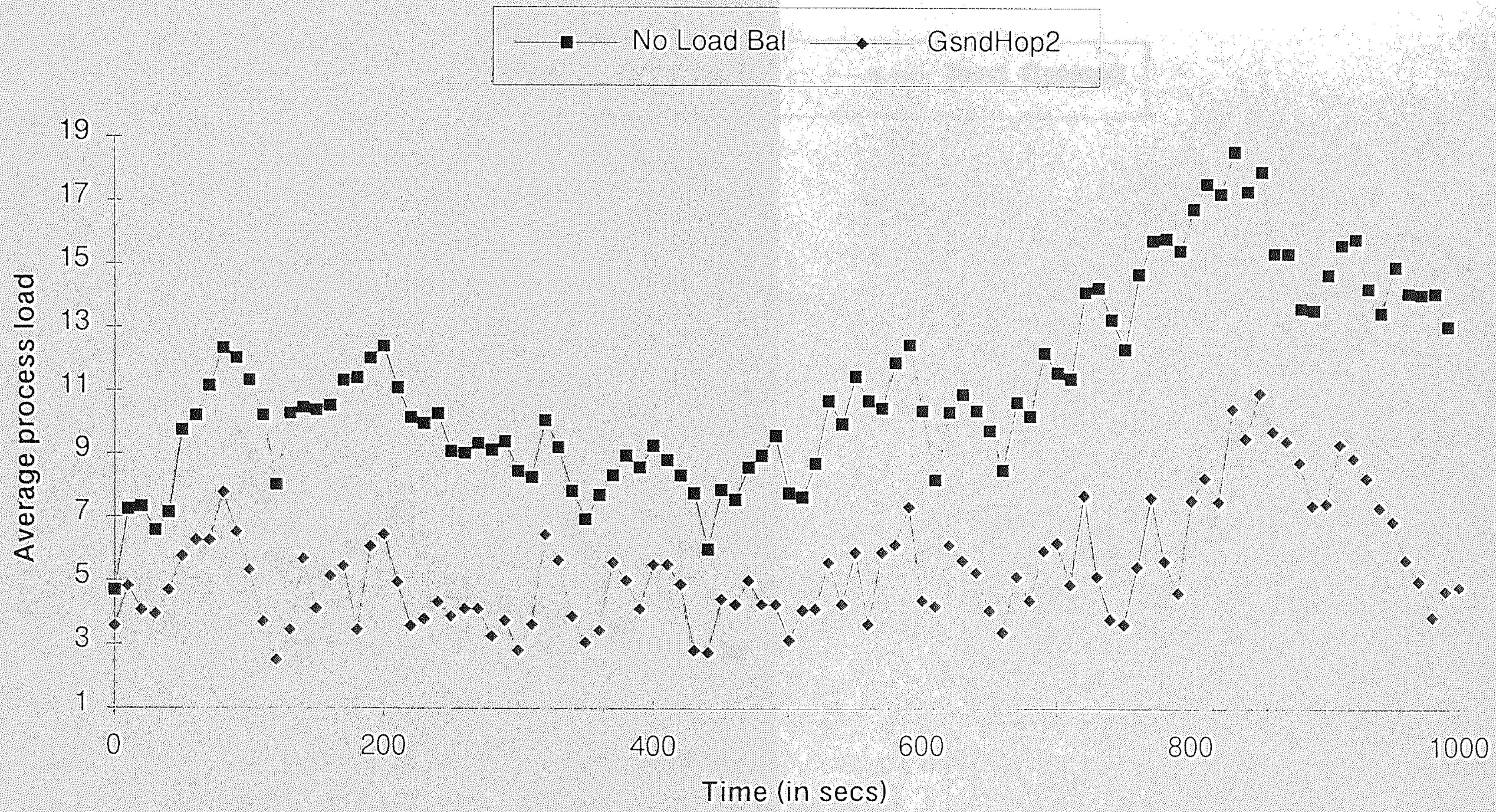


Figure B.2.23 No Load Bal vs Global Sender (Radius = 2 Hops) : Lightweight Process Workload Profile ($\varrho = 0.9$)

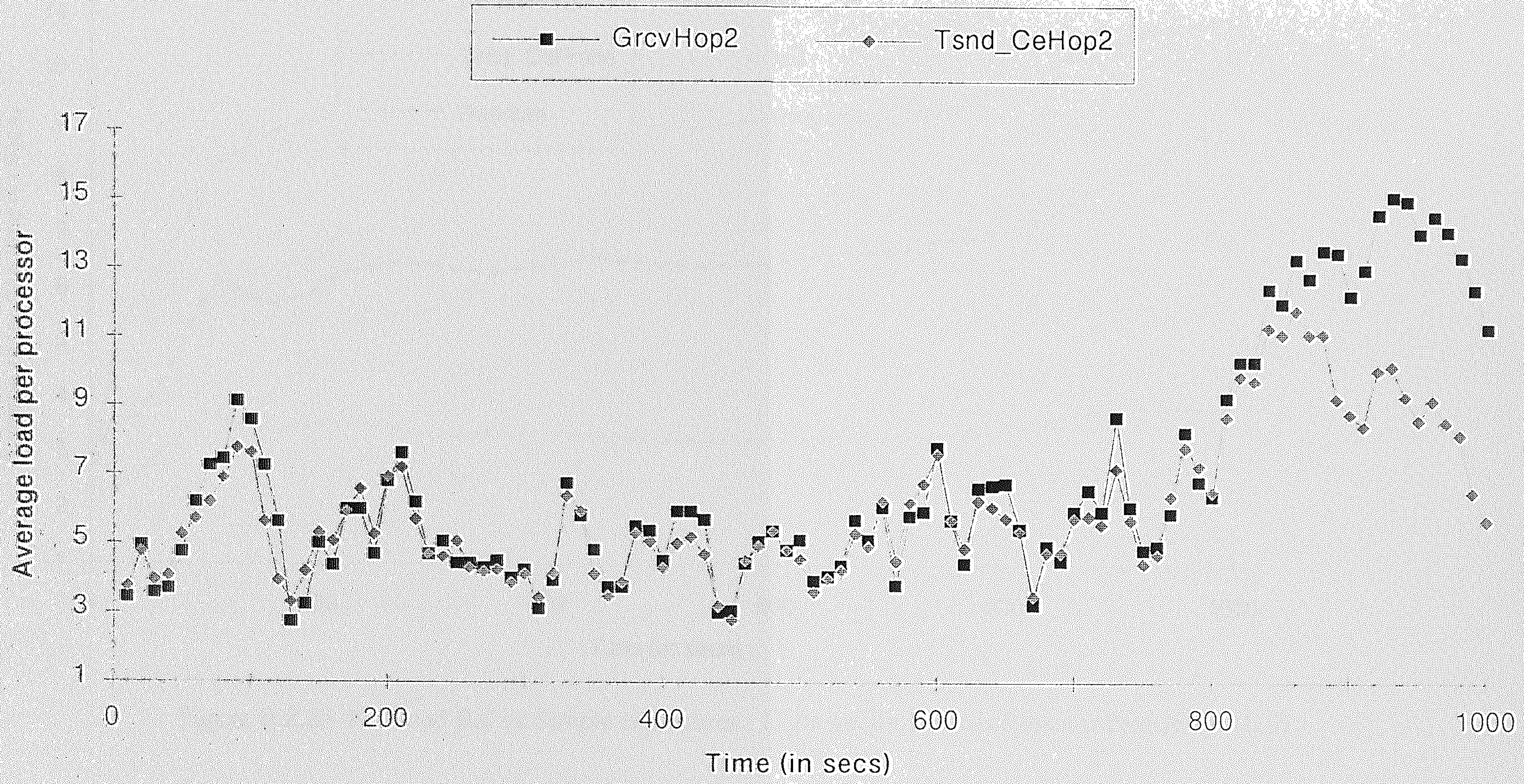


Figure B.2.24 Threshold Sender vs Global Receiver with Radius = 2 Hops : Lightweight Process Workload Profile ($\varrho = 0.9$)

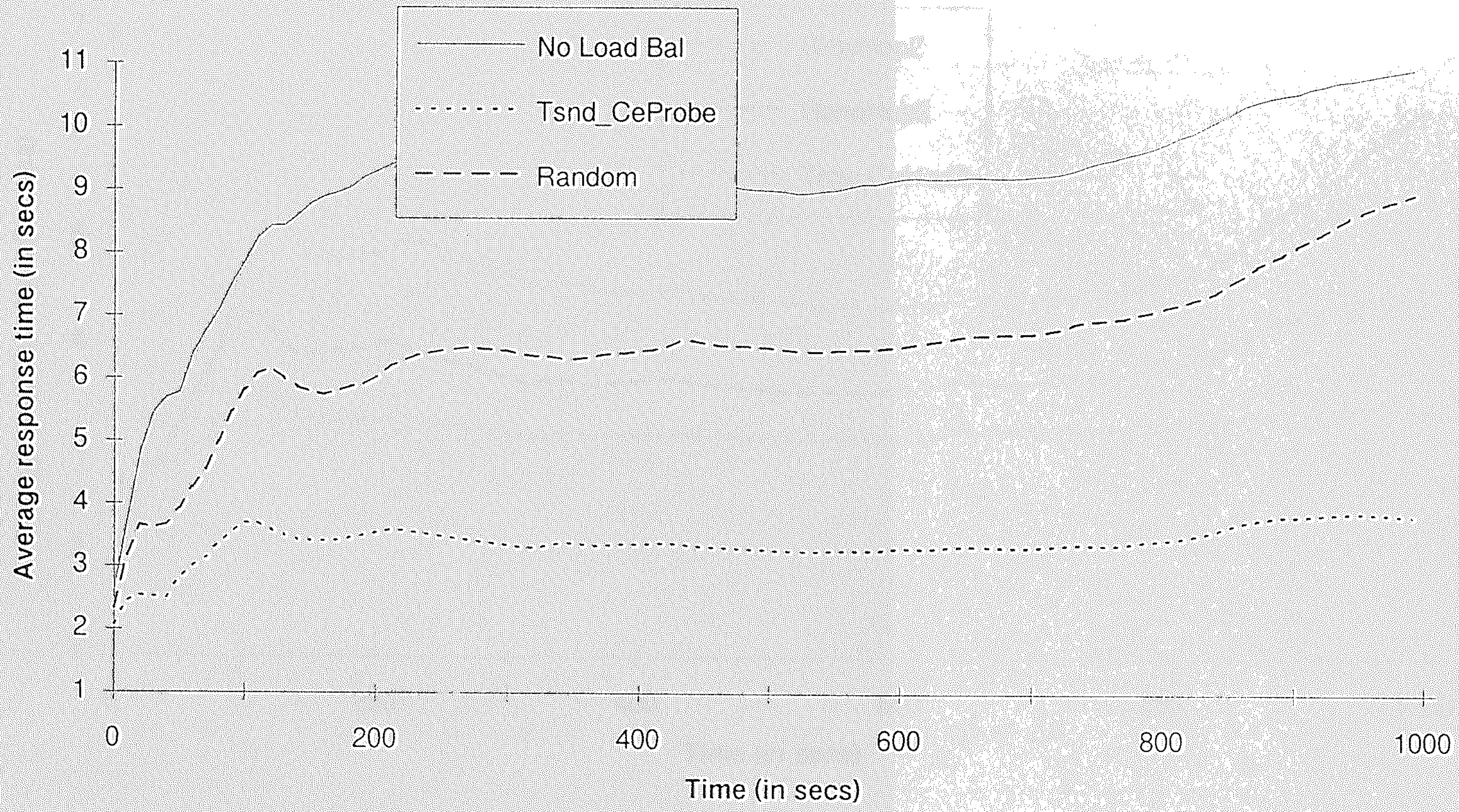


Figure B.2.25 No Load Bal vs Simple algorithms : Lightweight Process Runtime Profile ($\varrho = 0.9$)

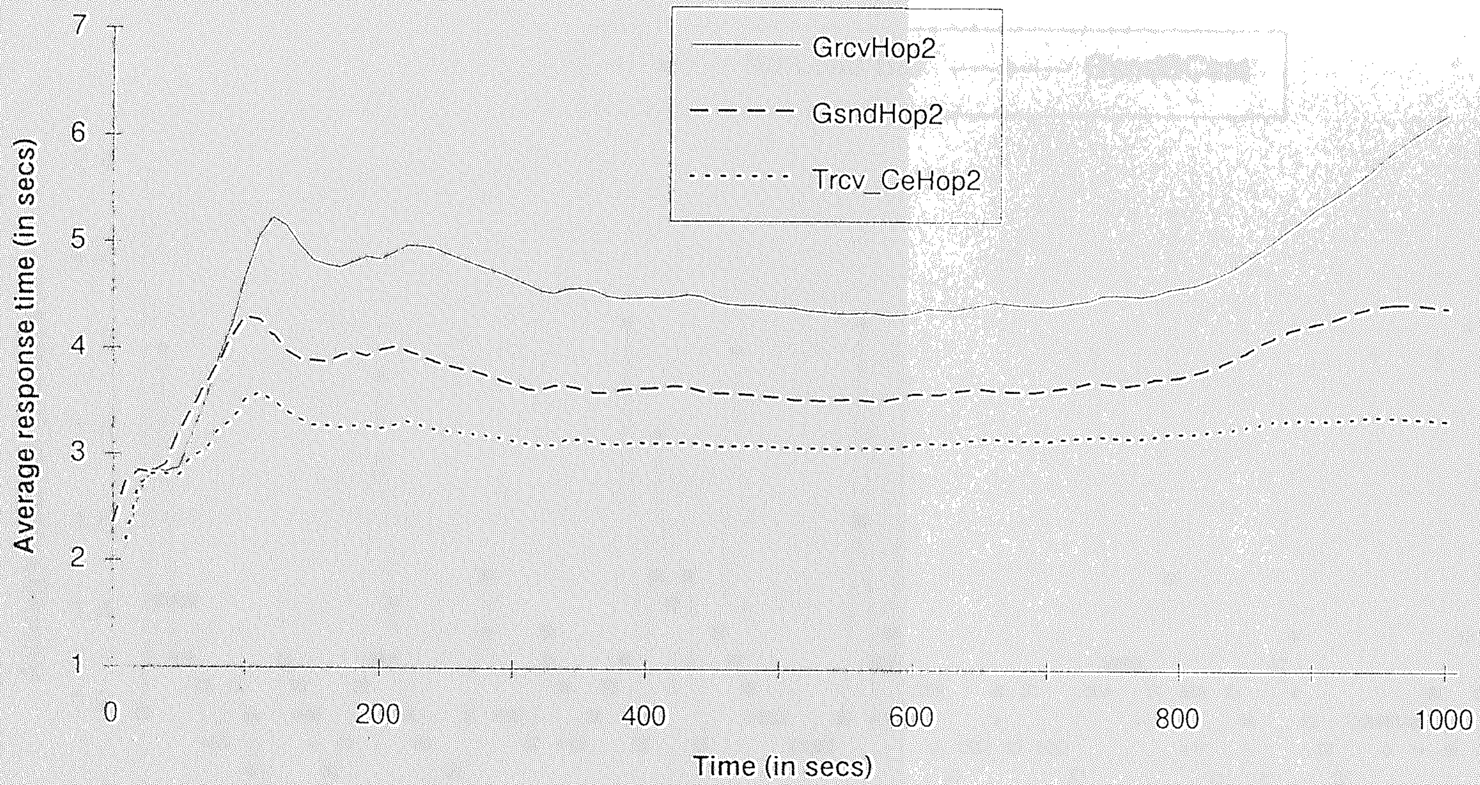


Figure B.2.26 Simple Adaptive vs Complex Global algorithms : Lightweight Process Runtime Profile ($\varrho = 0.9$)

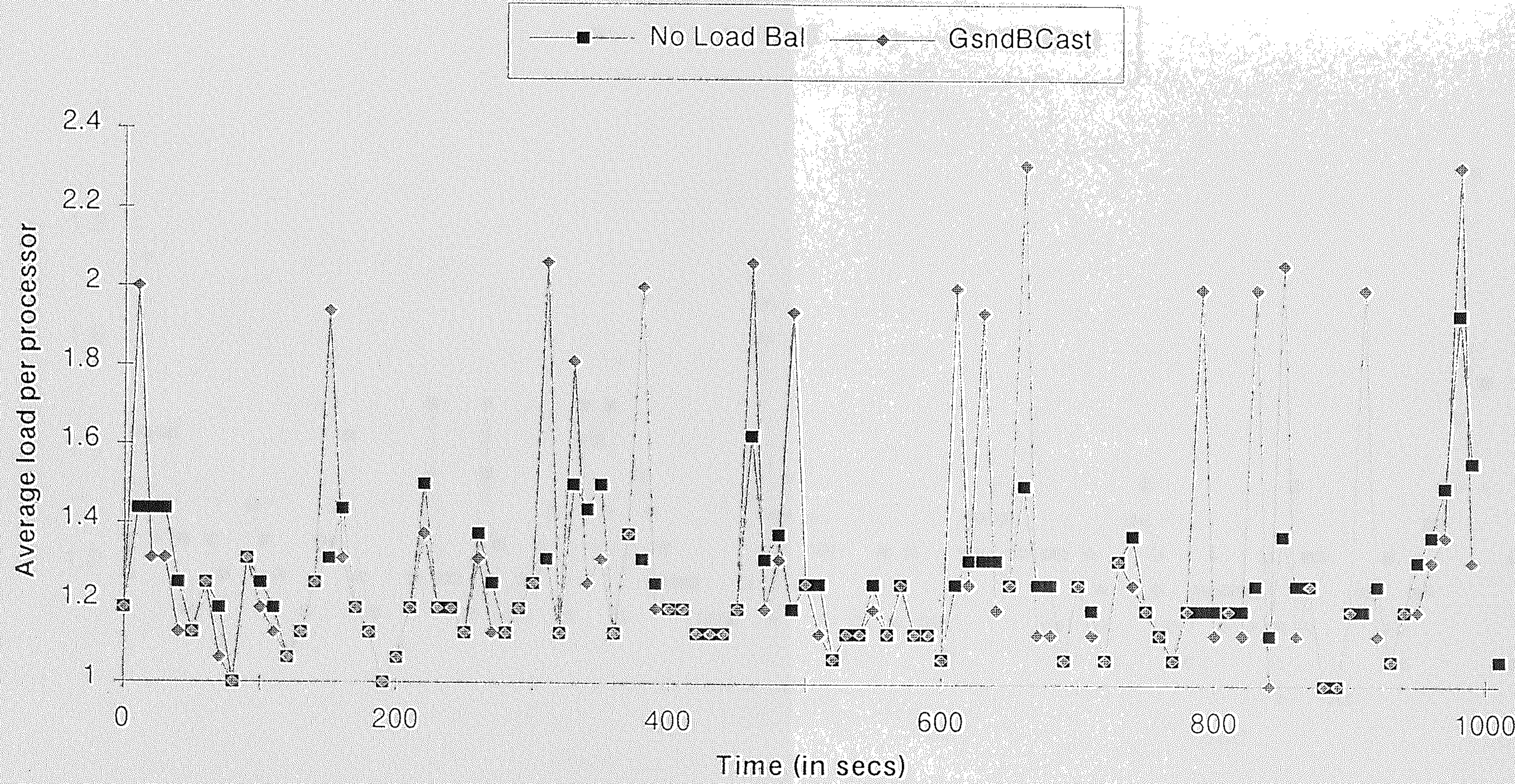


Figure B.2.27 No Load Bal vs Global (snd) Broadcast : Heavyweight Process Workload Profile ($\varrho = 0.2$)

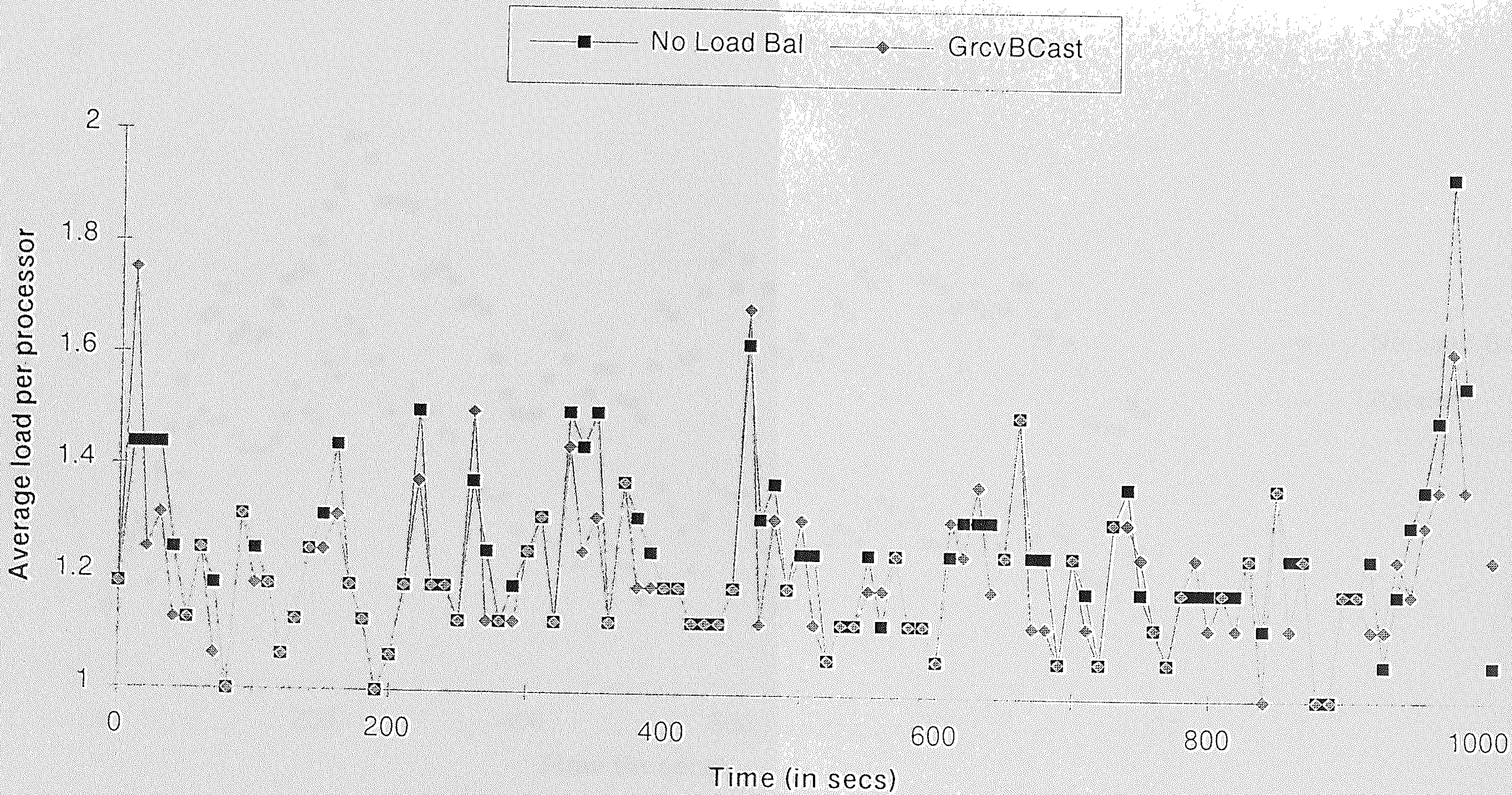


Figure B.2.28 No Load Bal vs Global (rcv) Broadcast : Heavyweight Process Workload Profile ($\varrho = 0.2$)

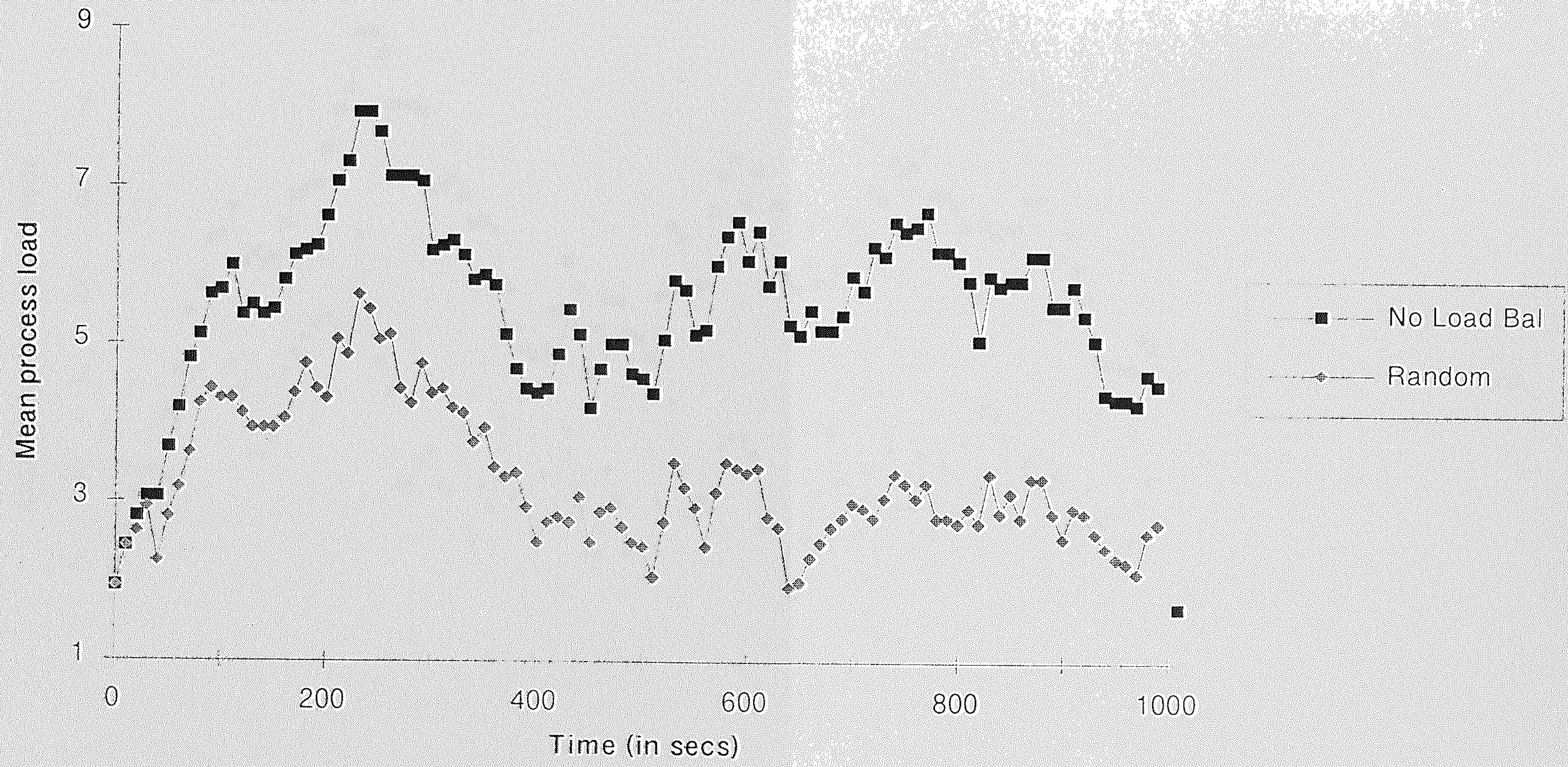


Figure B.2.29 No Load Bal vs Random : Heavyweight Process Workload Profile (ϱ = 0.8)

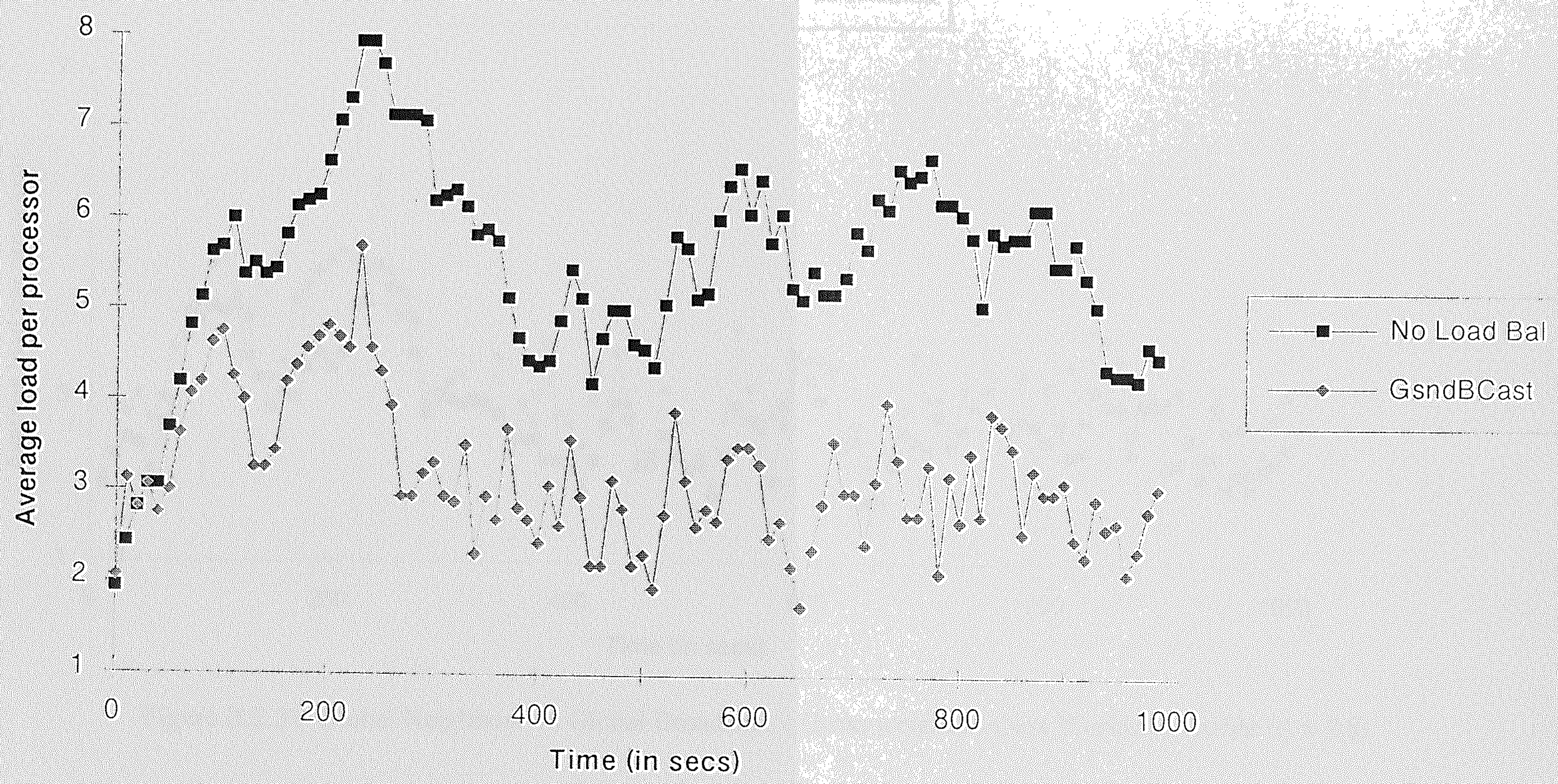


Figure B.2.30 No Load Bal vs Global (snd) Broadcast : Heavyweight Process Workload Profile ($\varrho = 0.8$)

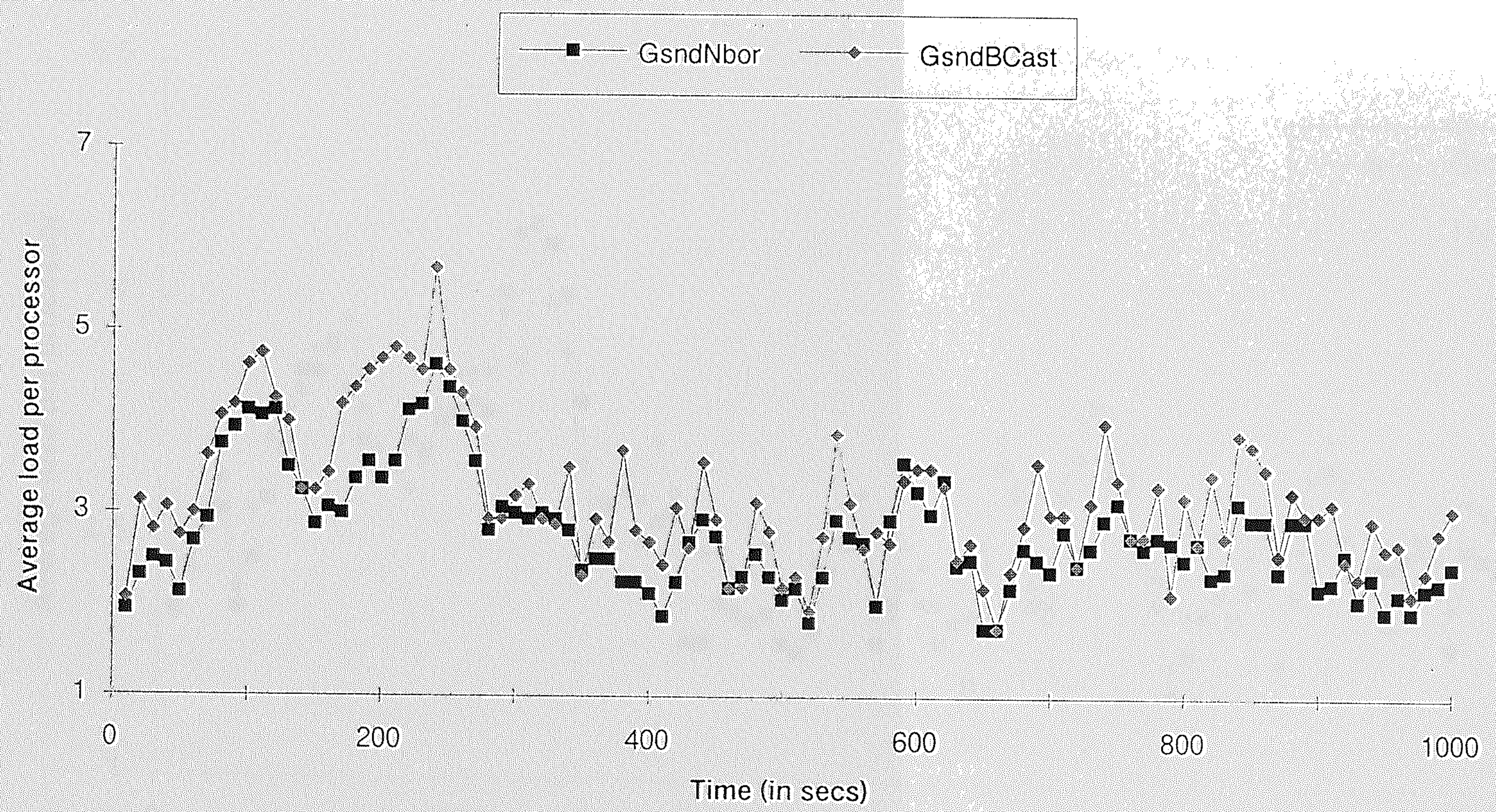


Figure B.2.31 Global Neighbour vs Global Broadcast : Heavyweight Process Workload Profile ($\rho = 0.8$)

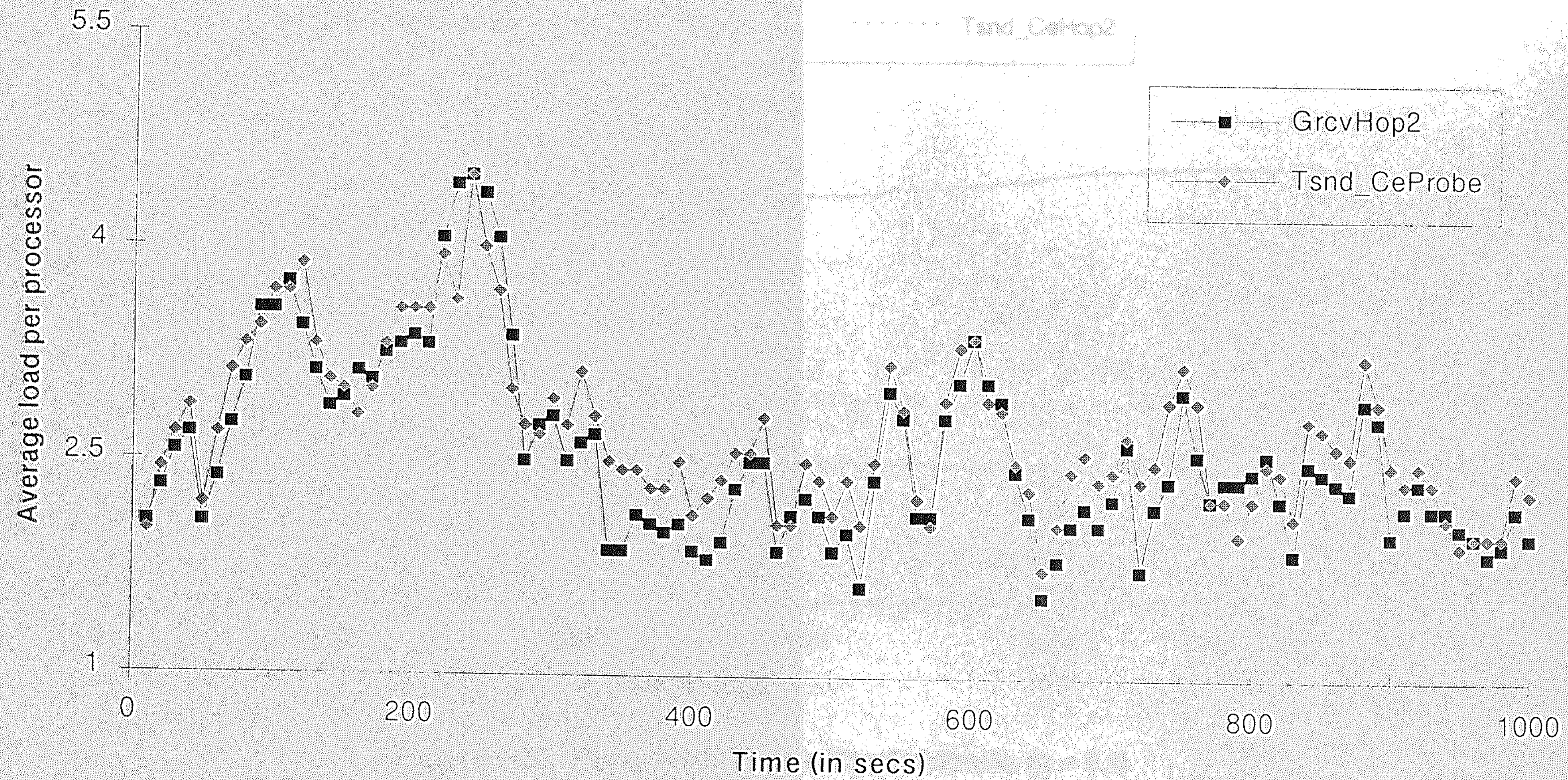


Figure B.2.32 Threshold (Cset) vs Global Sender (Hop = 2) : Heavyweight Process Workload Profile ($\varrho = 0.8$)

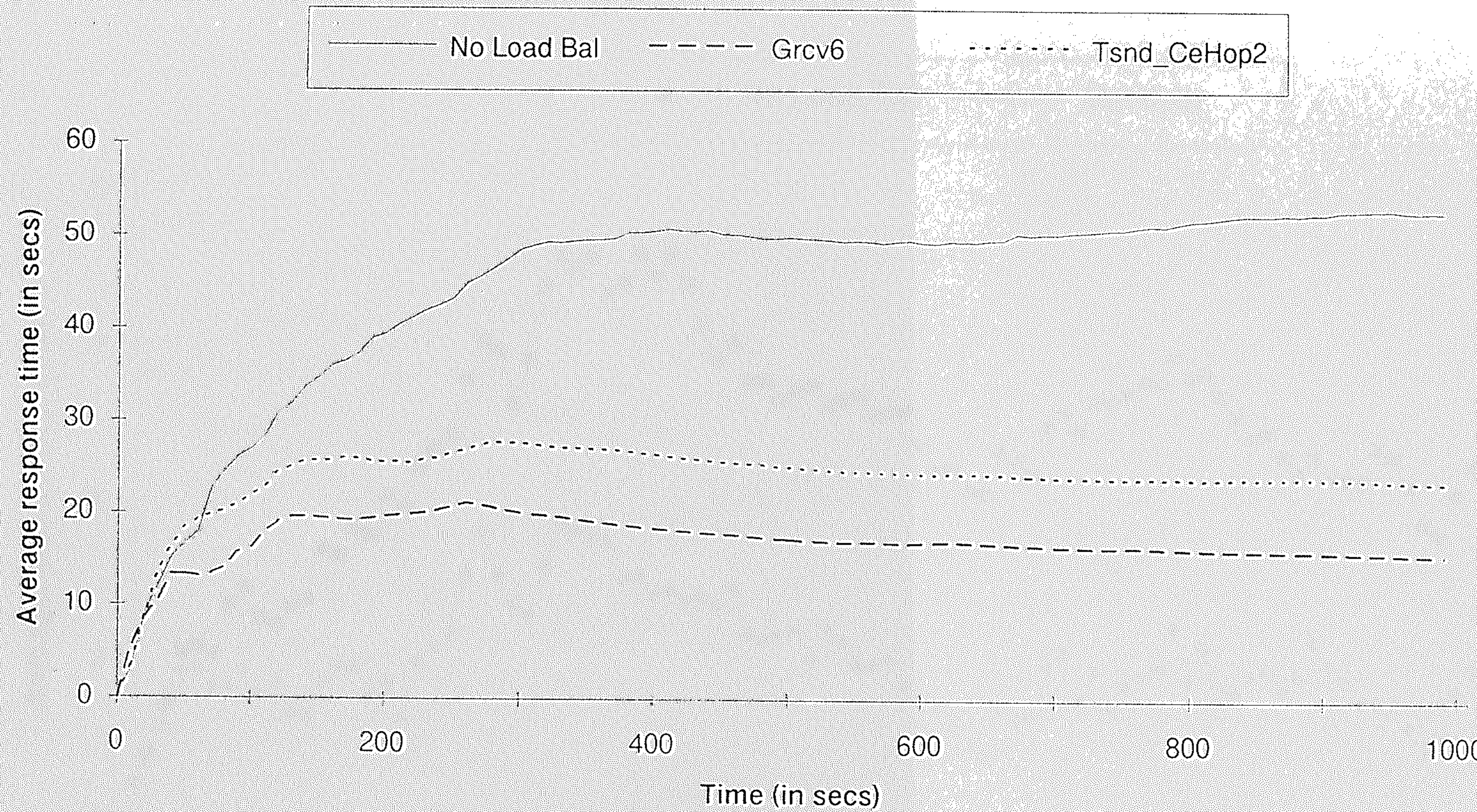


Figure B.2.33 Heavyweight Process Runtime Profile ($\varrho = 0.8$)

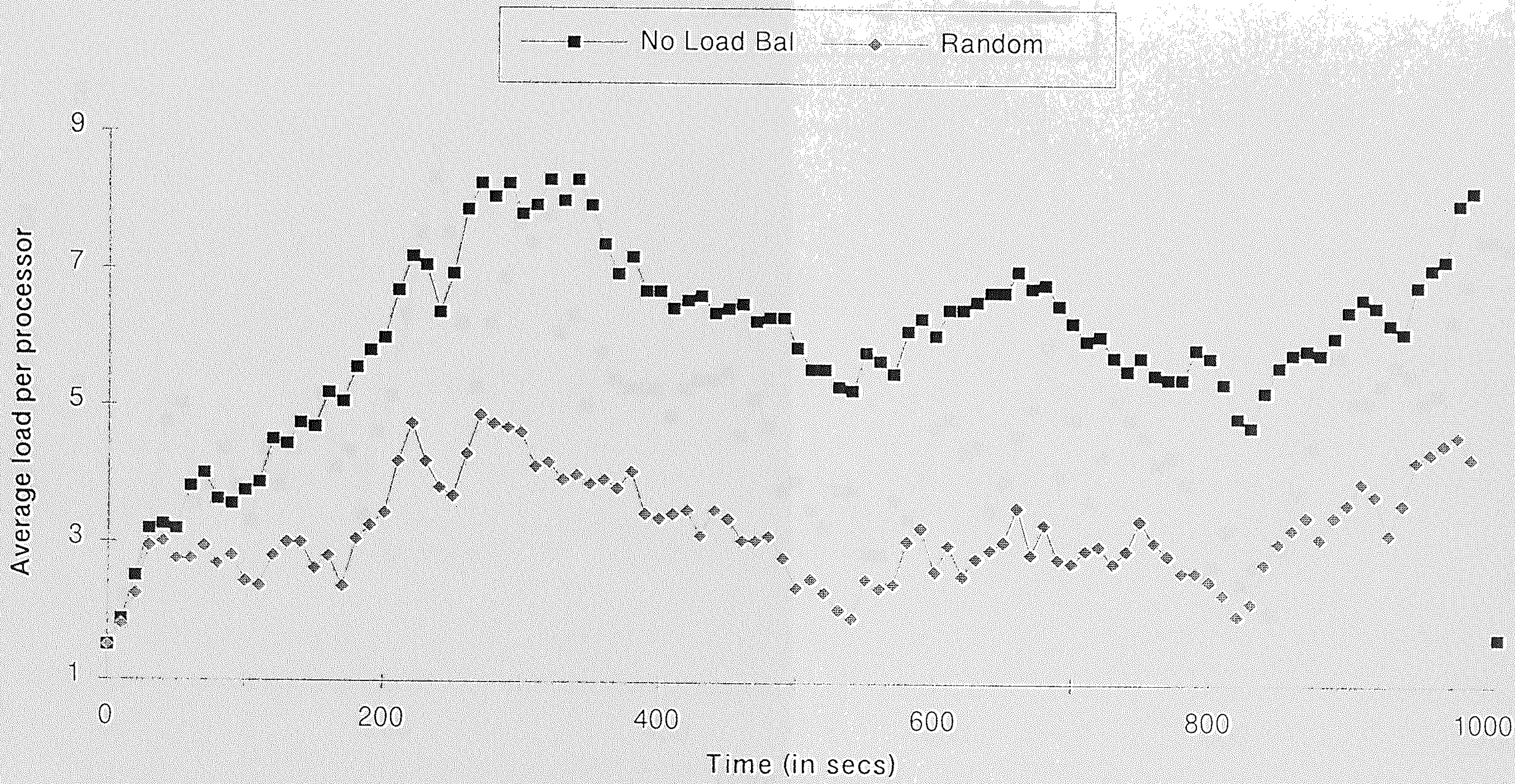


Figure B.2.34 No Load Bal vs Random : Heavyweight Process Workload Profile ($\varrho = 0.9$)

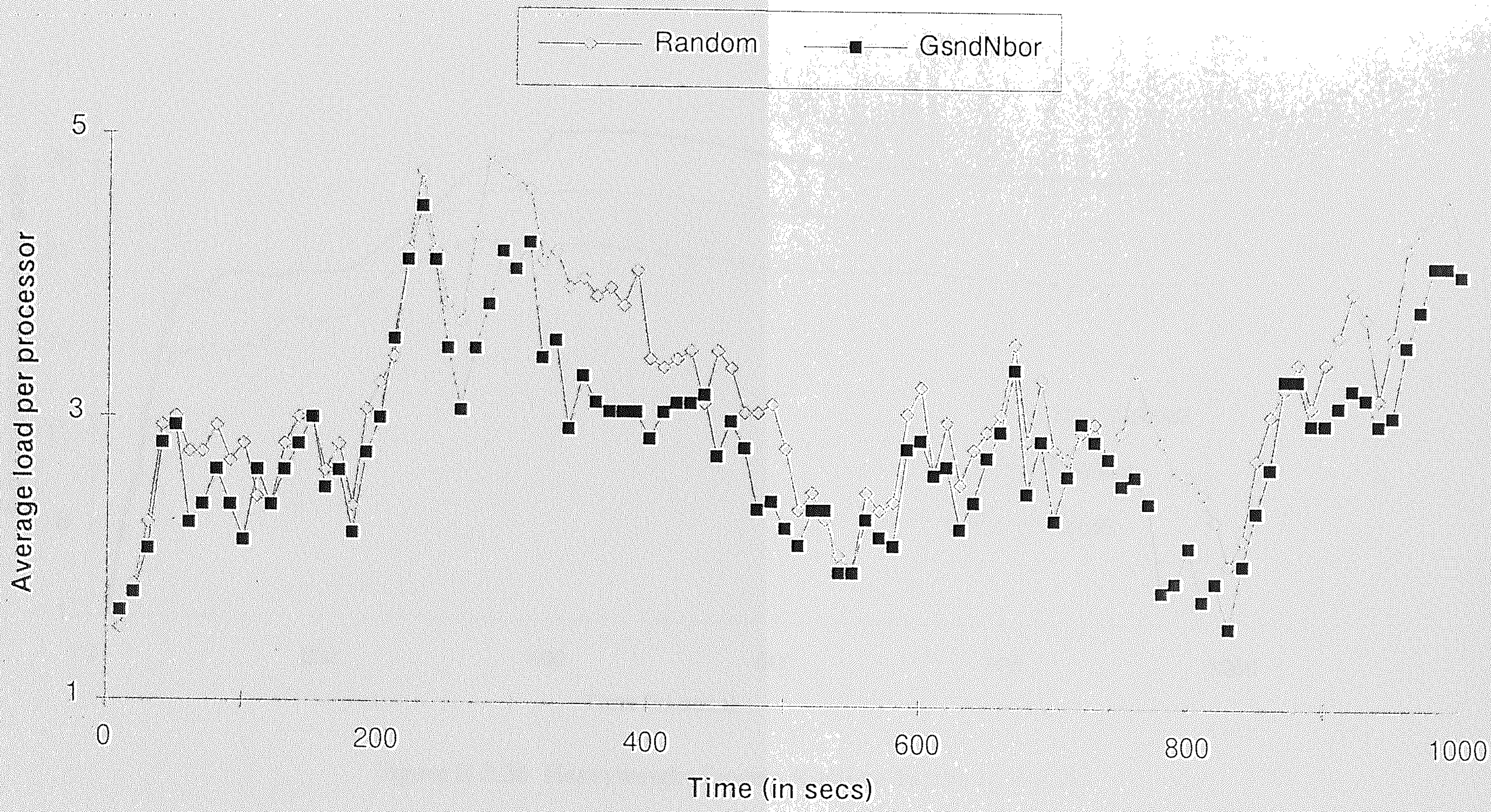


Figure B.2.35 Random vs Global (snd) Neighbour : Heavyweight Process Workload Profile ($\varrho = 0.9$)

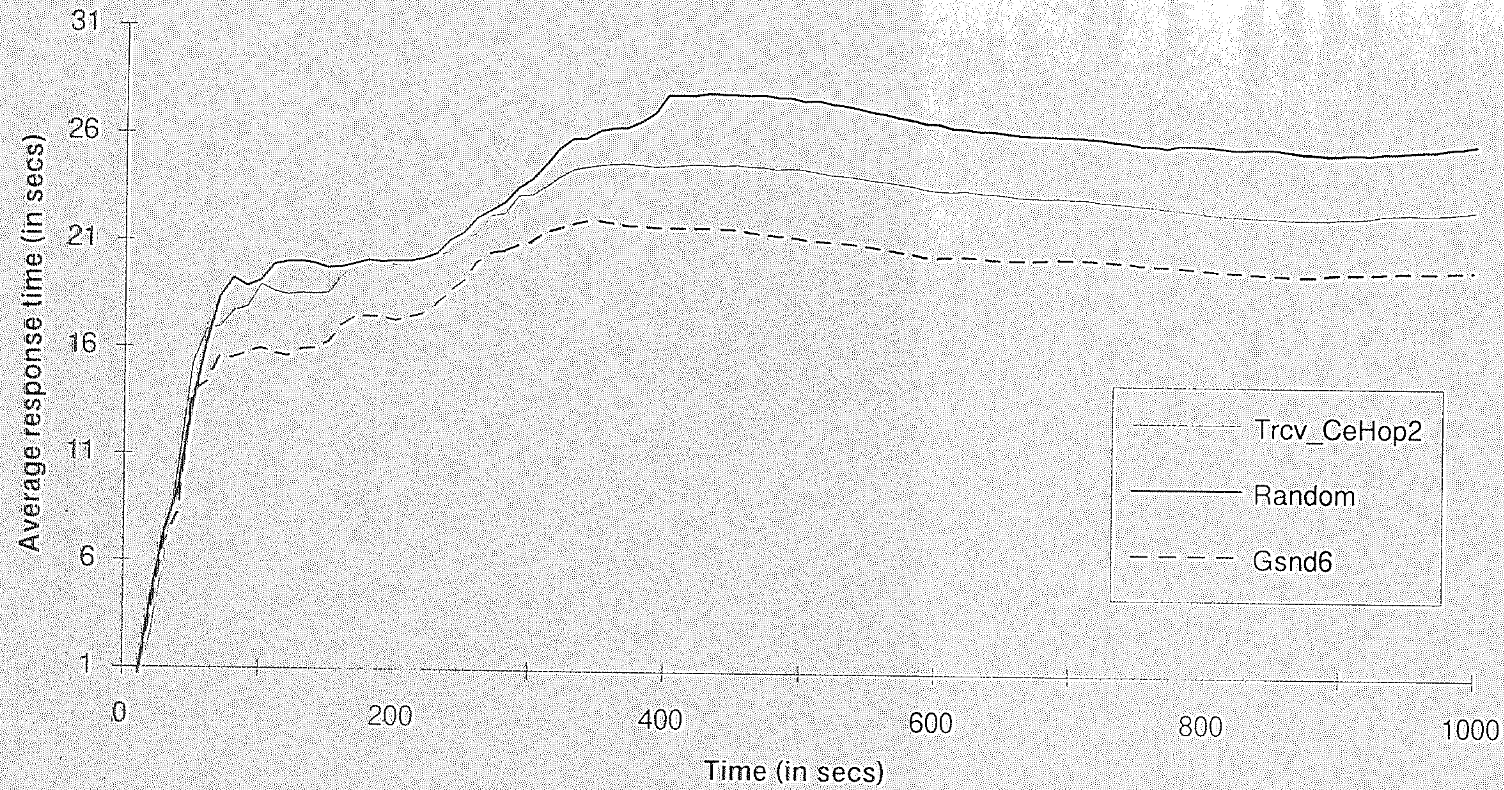


Figure B.2.36 Heavyweight Process Runtime Profile ($\varrho = 0.9$)

APPENDIX C

SOURCE CODE LISTINGS

C.1 DETAILED C++ DESIGN ABSTRACTION

The use of the C++ class constructs enabled the high-level design to be further refined towards an acceptable implementation. Thus, in the detailed design it was necessary to provide a more precise specification of the data structures, methods, and type of interaction required between objects.

The initial object identified was *type_Simulator* :

```
class type_Simulator
{
  private:
    type_System      *system;
    type_Monitor     *control_pe;

    char          *Get_Time(void);
    void           Display_Menu(int);
    void           run_simulation(void);
    int            menu_selection(int);
    void           Create_Config_File(int),
                    Create_Params_File(int),
                    write_intro_script();
    void           init_model_params (void);

  protected:
    static int            total_pe,
                          n_reached;
    static double         G_stop_time;


  public:
    type_Simulator(void);              // Constructor
    void           Interactive (void);
    void           Batch (void);
};
```

In the high-level design the objects *type_System* and *type_Monitor* were considered as derived classes. In so doing, objects of these types could also be viewed as simulated components providing access functions allowing them to operate in batch or interactive mode. However, this abstraction was considered invalid as these components may have little in common with *type_Simulator* when it is used as the base class. Further, an application would need to be developed (of type *type_Simulator*) to effect simulation using *type_System* and *type_Monitor* objects. Therefore, separate classes were adopted for the simulator, the system and the monitor objects. A simulator object can now define system and monitor objects such that a greater degree of control can be exercised over the size of the simulated system and the integrity of any results produced. This is ensured by using private definitions. Thus, if the model was to be represented as UNIX processes, a family group would be used where the simulator object, as parent, creates child processes "System" and "Monitor" and thereafter can order the dialogue with such components using appropriate sequences of IPC calls.

The following application defines a single simulator named *Model* that the application user can run in batch or interactive mode. The application cannot therefore, access directly the time, configuration, and event files of the model.

```
/* ============================================================ */
/*    main - Test harness for network processing functions.        */
/* ============================================================ */
main (void)
{
  int           batch_mode;
  type_Simulator   Model;

  batch_mode    = BATCH_RUN;
  if (!batch_mode)
    Model.Interactive();
  else
    Model.Batch();
} /* main */
```

The following section of code shows the implementation of the *Interactive* method of the simulator object. In particular, a menu-driven interface is used which allows the user to specify the characteristics of the system being simulated. This includes the total number of processors, the network topology, and the load balancing protocol to be used.

Also shown is the implementation of the private function *run_simulation* which activates one or more simulation runs in sequence. Of particular note is the call made to the public functions of the **System** object to *Configure, Boot,* and *Run* the system.

```
void type_Simulator::Interactive(void)
  { int           menu_id,
                  new_menu  = 1,
                  option;

    BOOLEAN         exit_loop;
    monitor_file  = stdout;
    do
    {/* select option and perform function                    */
     menu_id   = new_menu;
     option    = menu_selection(menu_id);
     switch (menu_id)
     {/* perform selected function                        */
       case 1: /*         */
             switch (option)
             {/* perform selected function                */
                case 1: new_menu  = 3;
                        break;
                case 2: new_menu  = 2;
                        break;
                case 3: break;
                case 4: run_simulation();
                        break;
               default: break;
             }/* switch */
             break;
       case 2: /*       */
             switch (option)
             {  case 1:  break;
                case 4: break;
               default: /* exit from program              */
                        new_menu = 1;
                        break;
             }/* switch */
             break;
       case 3:
             switch (option)
             {/* perform selected function                */
                case 1: break;
                case 4: break;
               default: /* exit from program              */
                        new_menu = 1;
                        break;
             }/* switch */
             break;
       default:/* exit from program                       */
             break;
     };/* switch */
     exit_loop = (menu_id == 1) && (option == 9);
    } while (!exit_loop);
  }
```

```
/* ============================================================ */
/*    run_simulation                                            */
/*                                                              */
/* ============================================================ */
void  type_Simulator::run_simulation (void)
{
  BOOLEAN        positive_response = FALSE;
  int              un_count  = 1,
                   prev_total = 0;
  fprintf (monitor_file,
        "\n *** SIMULATION COMMENCED ON %s \n\n", Get_Time());
  trace = fopen("main.trc", "a");
  Create_Config_File (BATCH_RUN);
  Create_Params_File (BATCH_RUN);
  configfile = fopen ("config.dat", "r");
  write_intro_script();
  while (!feof(configfile))
  {
   fscanf (configfile, "%d", &mesh_len);
   if (prev_total == total_pe)
   {
    fscanf (configfile, "%d", &positive_response);
   }
   else prev_total = total_pe;
   system = new type_System(total_pe);
   system->Configure();
   seedfile  = fopen("seedfile.dat", "r");
   paramfile  = fopen ("pramfile.dat", "r");
   fscanf (paramfile, "%f", &load_value);
   while (!feof(paramfile))
   {
    fprintf (monitor_file,
          "\nSIMULATION no.%d STARTED ON %s",
          run_count, Get_Time());
    init_model_params();
    system->Boot();
    system->Run();
    fprintf (monitor_file,
          "-----------------------------------------------\n\n\014");
    fflush (monitor_file);
    fflush (trace);
    run_count++;
    fscanf (paramfile, "%f", &load_value);
   }
   fclose (paramfile);
   fclose (seedfile);
   system->DisConnect();
   delete (system);
   write_intro_script();
  }
  fclose(configfile);
  fclose(trace);
}
```

The class *type_System* contain all functions and methods pertaining to a distributed system, namely, the network topology and the sets of nodes and their characteristics. The class definition is shown below.

```
class type_System

  private:

    void           Build_Route_Table(void);
    BOOLEAN        Balanced_Route (LOCAL_HOST *, int);
    BOOLEAN        Find_Links (LINK_ENTRY [],int, int);
    stack_entry   *Create_ItemQ(int, BOOLEAN, int, LOCAL_HOST *);
    BOOLEAN        Stack_Links (LOCAL_HOST *, int);
    void           Traverse_Net (LOCAL_HOST *);
    BOOLEAN        Push_Ltable (LOCAL_HOST *);

    void        Initialise_Netmodel(void);
    void        Init_Model_Params(void);
    void        Clear_Workspace (void);
    void        Build_Network(void),
                 System_Enable(void);

    float       Ave_Migration(void),
                Ave_Transmission(void),
                Mean_Difference(void),
                Variance(float),
                Process_Runtime(void);

  protected:

    NODE_ENTRY     *processor_table;
    BOOLEAN        Reachable (int, int, int, int);

  public:

    type_System(int total_pe = 9);
    int          Dump_Tables(void);
    void        Configure (void);
    void         Boot (void);
    void         Run (void);
    void         DisConnect(void);
};
```

It was at this point that crucial decisions had to be made about a system object and its components. Clearly, a system may be composed of one or more sub-systems. Thus, the individual nodes of a distributed system is a system in its own right, capable of autonomous operation. But, the key difference is their relationship with one another through a separate communication medium. Therefore, it was decided that objects of the type *type_System* class would be considered as "virtual" objects consisting of "real" and "imaginary" components such as the node and the network respectively.

Therefore, the *type_System* class was defined as containing private monitor and processor objects. The constructor for this class is set to a default system size of nine processors, and users of the system object can initiate configuration and reboot of the system as well as "core" (or table) dumps and power down. The following section of code shows the constructor, *Configure*, and *Run* methods for this class

```
type_System::type_System(int total_pe)
{
   processor_table = new NODE_ENTRY[total_pe];
}

void type_System::Configure(void)
{
   Build_Network();
}


void type_System::Run(void)
{
 type_Monitor  *control_pe;

 control_pe = new type_Monitor;
 while (!control_pe->Completed())
 {
   System_Enable ();
   control_pe->Update();
 }
 control_pe->Disable();
 delete (control_pe);
}
```

The constructor function creates an array of host objects according to the size of network required. The public access function *Configure* calls the private function *Build_Network* to establish the "logical" links between processor objects.

Note that the monitor object variable (control_pe) is created whenever a system is run and is used in a system object to detected the completion of a simulation run and to update both synchronisation and statistical information.

The class *type_Monitor* contains all the functions and structures relating to the monitor and control of the system. It is during the design of the monitor class that it became apparent that a monitor object would require access to information deemed to be private to a system such as the group or individual identity of a collection of hosts. In a Network Operating System this is not problematic as all hosts have a public identity known to the user of the system. Alternatively, it could be argued that this information be volunteered by the system object. However, to do so would result in a system object driving the monitor object in a master-slave relationship, thus reducing the effectiveness of the latter. Further, the interface to the system would become more complex in an attempt to make available to the monitor object all the information it requires. In addition, by making such functions public would result in "open access" to access methods for highly sensitive information.

The alternative would be to make *type_Monitor* a derived class of *type_System*. However, such an abstraction creates other problems as a monitor object would itself be a distributed system inheriting the methods and structures of the *type_System* class. Further, the simplicity and autonomy of the *type_Monitor* class would be lost.

```
class type_Monitor
{
  private:
    void          Initialise_Nodes(void),
                  Reset_Reached (void);
    float         RTime_Convergence(type_System *, int);
    BOOLEAN       Stable_System(type_System *);

    void          Ave_Compute_or_Display(type_System *, BOOLEAN),
                  Final_Compute(type_System *);
  public:
    type_Monitor(void);          // constructor
    void          Update (void);
    BOOLEAN         Completed (void);
    void          Disable(void);
};
```

It is clear that a monitor object is part of a system and is declared as a private dynamic object to reflect this. Such a declaration should allow the said object privileged access unavailable to other users of the system. However, the C++ implementation does not operate in this manner, but require that each class define who its "friends" are and, as such, have access to private methods and data. Thus, the *type_System* class was refined as follows:

```
class type_System
{
    friend void    type_Monitor::Ave_Compute_or_Display(type_System *,
                                  BOOLEAN);
    friend float  type_Monitor::RTime_Convergence(type_System *, int);

    friend BOOLEAN type_Monitor::Stable_System(type_System *);
    friend void    type_Monitor::Final_Compute(type_System *);

  private:
    ........
    ........
}
```

In order that the side effects of global availability be minimised, the above example declared specific methods of the ***type_Monitor*** class to be friends rather than the complete class. However, this is not ideal as the system object must be identified and passed as a parameteer to the named method of the *type_Monitor* class.

The class *type_LocalHost* was perhaps the most important and difficult class to conceptualise during the design stage, namely because some of its characteristics are inherited from the type of simulation required and the *type_System* class. In a loosely-coupled system, a local host consist of its own operating system and user application processes. Each host will also have its own set of timers and communication "cards".

```
class type_LocalHost
{
  friend class   type_System;

 private:
  short int             node_id;
  unsigned short        xsubi[3];

  LINK_ENTRY            *links;
  LOAD_ENTRY            cSet[CSET_SIZE];

  int           load,
                threshold,
                n_tx, n_rx,
                n_migrates,
                n_immigs,
                n_deaths,
                n_local_procs,
                n_active_local_procs,
                n_virtual_procs;
  float         wt;

  BOOLEAN           high_t_set,
                    low_t_set,
                    pwait_set,
                    rplimit;

  ROUTE_ENTRY       route_table[TABLE_SIZE];

  double            nxt_arr_time,
                    cumul_exist_time;

  type_OPSystem     *osVX;

  short int         buffer_no;
  char              event_fname[16];

  FILE              *job_file,
                    *trace_file;

  void              do_processing (void); /*run */
  void              HostNet_Setup(int, int);
  int               Ave_Call(void);
  BOOLEAN           Update_Links (int, STACK_ENTRY *);
  int               Dump_MyLinks (int);

 public:
  type_LocalHost(int id = 0);
  void              Reset_Pe(void);
  BOOLEAN           Suspend_Pe(void);
  void              Run_Pe(void);
};
```

It is notable that the complete class *type_System* is declared as a "friend" of the *type_LocalHost* making the complete components of the latter accessible to the former. In terms of the software of the system, the operating system and user processes were initially considered as independent but related entities. Again, their interrelationship present problems in terms of the proliferation and control of "friends". A further refinement is to consider the user processes as an integral component of a "virtual" software system, namely the operating system. Therefore, the variable *osVX* represent the operating system interface access to the applications and resources of the local host. The public interface to a local host object consist of three basic functions to reset, suspend and run a local host object. The constructor function for a local host is shown below:

```
/* ============================================================ */
/*                                                              */
/*   type_LocalHost::T Y P E _ L O C A L H O S T               */
/*                                                              */
/* ============================================================ */
type_LocalHost::type_LocalHost(int id)
{ /* implementation */

    int     idx;

    /* Construct data node */
    node_id       = id;
    links         = new LINK_ENTRY[total_pe];

    for (idx = 0; idx < total_pe; idx++)
    {
      links[idx].node_id  = EMPTY;
      links[idx].hops     = total_pe;
    }

    for (idx = 0; idx < 4; idx++)
    {
      route_table[idx].node_ptr = NULL;
      route_table[idx].node_id  = EMPTY;
    }
}
```

This function creates and initialises the links and route tables required for communication purposes. This function also creates (or initialise the operating system object). One can envisage a UNIX shell process being activated at this stage. The implementation of the public functions *Run_Pe* and *Suspend_Pe* can then make calls using the public interface of the operating system object. The implementation of these methods follow:

```
void type_LocalHost::Run_Pe(void)
{

  osVX->Msg_Processor();
  if (!Suspend_Pe())
    osVX->Event_Processor();
}
```

```
BOOLEAN type_LocalHost::Suspend_Pe(void)
{
  if (osVX->Time_Check() >= G_stop_time)
  {
    if (!processor_table[node_id].await_sync)
    {
      n_reached++;
      processor_table[node_id].await_sync = TRUE;
    }
    return TRUE;
  }
  else
    return FALSE;
}
```

It is worth noting that a local host will require access to the processor table of the *type_System* class which was achieved by defining "friendships". The following code represents the method to set up the network address tables for the local host.

```
/* ============================================================== */
/*                                                                */
/*     type_LocalHost::H O S T N E T _ S E T U P                 */
/*                                                                */
/* ============================================================== */
void type_LocalHost::HostNet_Setup(int lidx, int link_id)
{
  route_table[lidx].node_id = link_id;
  route_table[lidx].node_ptr = processor_table[link_id].node_adr;
  links[link_id].node_id = link_id;
  links[link_id].hops    = 1;
}
```

The operating system is generally regarded as protected software. However, having decided to make user processes a component of the "virtual" operating system object, the issue of kernel and user process characteristics is significant. Whilst such processes may share a common base class, the interrelationship of the kernel processes, from which an operating system is derived, and their interrelationship with user processes created further implementation complexity using the C++ mechanisms. Therefore, the class definition given below identifies the main kernel processes which include load balancing, local process scheduling, message handling, and time management. A user process object is also a private component accessible using the variable *utask*.

```
class type_OPSystem
{
 private:
    type_LBProcess        *lba;
    type_MSGProcess       *msg_handler;
    class type_SCHEDProcess *scheduler;
    type_TIMProcess    *clock;

    type_USRProcess    *utask;

    float      Dyn_Calc_Ave_Load(type_TIMProcess *);
    float      Calc_Ave_Load(type_TIMProcess *);

 public:
  double        Time_Check(void);

  void       Msg_Processor(void),
             Event_Processor(void),
             Restart(void);
             Suspend_Kernel(void);
};
```

In general, user processes and kernel processes have common characteristics. Therefore, a base class *type_AProcess* was defined. Each process has a private process control block, and communicating set information. Likewise, the basic public function available is the return of the process identification number assigned by the operating system. This class definition is given below:

```
class type_AProcess
{
 private:
  process_cntrl_block      *pcb;
  void                  Update_Cset(MSG_FRAME *, int),
                        Delete_Cset (int);
 public:
  int              Get_Pid (void);
};
```

One can then define derived classes *type_KNProcess* and *type_USRProcess* using the above base class. However, the primary difference between both process types rests in their priority, the privileges enjoyed, and their behaviour. One had considered defining processes such as the scheduler and message handler as independent *type_KNProcess* objects, but their interdependency meant that the implementation complexity could be minimised by viewing such processes as private methods belonging to a kernel object. The class template outline is given below:

```
class type_KNProcess:public type_AProcess
{
  private:

// Process Priority store and functions

     void  Exit_Process (MSG_FRAME);
     void schedule_process (LOCAL_HOST *);
     ...............
  protected:

  int      load,
           threshold,
           n_migrates,
           n_immigs,
           n_deaths,
           n_local_procs,
           n_active_local_procs,
           n_virtual_procs;
  public:
  BOOLEAN        TimeOut_Handler(void);
  void           Schedule_Process (LOCAL_HOST *node_P);
  .....
};
```

Another problem presented by this definition is that the methods of the kernel object must be informed of the associated Host object by means of parameters. Thus, the separate definition of kernel processes as objects of the local host operating system addressed this problem.

The class *type_LBProcess* defined the load balancing methods available. The public interface included service categories for load balancing activity such as sender or receiver-initiated algorithms. The detailed implementation of these algorithms were private to the load balancing process object. The class definition is given below:

```
class type_LBProcess:public type_KNProcess
{
  private:
     random_policy ( );
     threshold_policy ( );
     thresh_Bcast ( );
     reverse_policy ( int);
     Rev_avg_rcv ( int);
     Rev_avg_snd( int);
     global_avg_snd(int);
     global_avg_rcv(void);
  public:
    receiver_initiated (void);
    sender_initiated ();
    Communicating_Set (int);
    Virtual_Sender (void);
    Virtual_Receiver (void);
};
```

Given the potential number of user processes that can exist in a large system under heavy load conditions, the *type_USRProcess* class was optimised such that every process object that would created in the system did not end up with a set of methods to schedule, create, update and edit itself. Thus, the following class specification consist of a queue of user process control blocks, but possessing the functionality required to emulate a user process:

```
class type_USRProcess:public type_AProcess
{

   friend void type_TIMProcess
        ::Time_Update (type_USRProcess *, int, int);

  private:

   usr_PCB          *current,
                    *firstjb,
                    *lastjb;

  public:
   void       Restart(void);
   usr_PCB *     Add_Process (int, usr_PCB *);
   usr_PCB *     Find_Process (usr_PCB *);
   usr_PCB *     Remove_Process (int);
   usr_PCB *     Create_New_Process(int);
};
```

From this definition it is notable that the time_update function of the type_TIMProcess objects needs access to certain components of a user process and, in addition the aggregate class have all the functions that would be required by each user process to manage itself. The process control block type definition (for usr_PCB) was as follows:

```
struct process_cntrl_block
{
    int             upid;
    int             n_probes;
    int             residency;
    double          exec_time;
    int             exec_here_time;
    int             residency_time;
    double          exist_time;
    int             orig_mc,
                    preferred_mc;
    BOOLEAN         schedulable,
                    blocked,
                    finished,
                    mcs_probed[6];
    double          timeout;
    int             itype;
    process_cntrl_block     *nextP;
};
```

A similar strategy was also adopted for messages of the system. Thus, as an object each message should be able to manage itself in terms of its construction, send and receive, and any resulting queue positions. Again, the overheads under heavy system load would make such a design impractical. Therefore, the class *type_MSGProcess* was defined with a message queue data structure and the methods required to act upon each element of the structure. These definitions are given below:

```
struct comm_packet
{
    double          time_stamp;
    int             sender_id,
                    dest_id,
                    distance,
                    total_bytes,
                    type,
                    service_no;
    double          ldvalue;
    BOOLEAN         ack_reply;
    usr_PCB         *mess_data;
    comm_packet     *nextM;
};
```

```
class type_MSGProcess
{
 private:
  MESS_QUEUE        queue;
  MSG_FRAME        *msg_buffer;
  void             Exit_Message (MSG_FRAME *);
 public:
  BOOLEAN           Receive(void);
  void             Transmit(void),
                   Restart(void),
                   Process_Message(void);
};
```

The class type_SCHEDProcess, given below identifies the a timer object method as "friendly" and, in the public interface, the scheduling discipline to be operated. The class definition is given below:

```
class type_SCHEDProcess:public type_KNProcess
{
  friend void type_TIMProcess::Next_Quantum(void);

  private:
   ....
  public:
   usr_PCB * Longest_Runner(void),
           * Migratable_Task(void);
   .......
};
```

Likewise, the class *type_TIMProcess* definition is given below. It is notable that all time-dependent and related functions are included in this template. These include timeout message handlers. The code displays some of the weaknesses of a C++ implementention using "friends" and object parameters for autonomous but related classes.

```
class type_TIMProcess:public type_KNProcess
{
  friend float type_OPSystem::Dyn_Calc_Ave_Load(type_TIMProcess *);
  friend float type_OPSystem::Calc_Ave_Load(type_TIMProcess *);

  private:

  double          sys_real_time,
                  next_elapsed_second,
                  last_perf_dump_time;

  QTUM_ENTRY           quanta[NQUANTA];
  int                  quanta_idx,
                       OSoverhead;
  TMER_QUEUE           timq;

    void          Wrq_Timout_Msg(void),
                  Hi_Timeout_Msg(void),
                  Avgld_Tout_Msg(void),
                  Low_Timeout_Msg(void);

    void          Select_Next_Timeout(void);
    void          Service_Routine(void);

    void          Quanta_Update(int, int);
    void          Next_Quantum(void),
                  Add_To_Quantum(int, int);
 public:
    BOOLEAN       Cancel_Timer (int, usr_PCB *);
    BOOLEAN       TimeOut_Handler(void);
    void          Restart_Clock(int , double, usr_PCB *);
    void          Reset(void),
                  Time_Update (type_USRProcess *, int, int);
   double         Display_Time(void)
                  {return sys_real_time;}
};
```

Whilst it is good object-oriented programming practice to avoid using "friends" and optimising the behaviour of objects, attempting the implementation of the model in C++ was very limiting. The proliferation of "friends" and the passing of objects as parameters increased the complexity of the interface between objects, and reduced the potential size of the system that could actually be modelled. Therefore, the high level design and the methods developed were grouped into five ANSI C source files which elliminated "friendships" and provided for a more efficient and meaningful implementation. The ANSI C listing for the model is given in the next section. The files are:

Kernel_Module.c	*type_AKNProcess*
Msg_Module.c	*type_MSGProcess*
Main_Module.c	*type_Simulator*
	type_System
Initb_Module.c	*type_System*
Stat_Module.c	*type_Monitor*

The design exercise was not fruitless as it enabled an early identification of all the methods that would be required in the implementation. Given a more flexible object oriented implementation language that retain the links between high-level abstractions and detailed implementation, the software construction stage for the model would have contained the actual amount of software developed for the experiments conducted.

The remaining portion of this chapter gives a sample of some of the implementation methods prototyped for the classes specified.

```
void type_TIMProcess::Reset(void)
{
   sys_real_time    = 0.0;
   last_perf_dump_time  = 0.0;
   next_elapsed_second  = 1000000.0;

   quanta_idx      = 0;
   OSoverhead      = 0;
   for (int idx2 = 0; idx2 < NQUANTA; idx2++)
    {
      quanta[idx2].actual_load  = N_SYS_PROCS;
      quanta[idx2].virtual_load = 0;
      quanta[idx2].OSportion   = 0;
      quanta[idx2].used        = 0;
    }

   timq.minP   = NULL;
   timq.itype   = 0;
   timq.mintime = 0;
}
```

```
void type_USRProcess::Restart(void)
{
    current   = NULL;
    firstjb   = NULL;
    lastjb    = NULL;
}

void type_MSGProcess::Restart(void)
{
   queue.tail  = NULL;
   queue.head  = NULL;
}

void type_LocalHost::Run_Pe(void)
{

  osVX->Msg_Processor();
  if (!Suspend_Pe())
    osVX->Event_Processor();
}
```

```
BOOLEAN type_LocalHost::Suspend_Pe(void)
{
  if (osVX->Time_Check() >= G_stop_time)
  {
    if (!processor_table[node_id].await_sync)
    {
      n_reached++;
      processor_table[node_id].await_sync = TRUE;
    }
    return TRUE;
  }
  else
    return FALSE;
}

void type_OPSystem::Msg_Processor ()
{
 BOOLEAN rcv;
 do
 {
  rcv = msg_handler->Receive();
  msg_handler->Process_Message();
  clock->TimeOut_Handler();
  if (!rcv)
    break;
 } while (1);
}
```

C.2 SHELL SCRIPTS FOR DATA MANAGEMENT

This section contain shell scripts that were written to convert the trace output of the simulation model into a format suitable for performing statistical analysis. Thus, the following output is typical for a collection of simulation runs:

```
*** SIMULATION COMMENCED ON Tue Jun  1 17:26:23 1993

Load Balancing algorithm invoked every 20 - 50 milliseconds!

SIMULATION no.1 STARTED ON Tue Jun  1 17:26:23 1993

No. of Processors =       9      Load value       = 0.80
Total No. of jobs =  1000000     Ld Balancing Alg. = Global Avg Nbor Policy

Threshold Level  =    2.00       Distance        = 4
Convergence to < 2.00%

  TIME      VAR.     LOAD    DIFF.    RTime    %Conv    MIG.      Tx      %JOB
  ----      ----     ----    -----    -----    -----    ----      --      ----
1800.00    0.7850   3.9543  2.4406   3.3098    7.48    0.8988   36.0651   1.29
3600.00    0.7792   3.9503  2.4364   3.1571    2.85    0.8968   35.8049   2.58
*** TERMINATING CONDITION : CONVERGENCE @2.00%***

SIMULATION COMPLETED ON Tue Jun  1 17:32:24 1993
------------------------------------------------
```

```
SIMULATION no.2 STARTED ON Tue Jun  1 17:32:24 1993

No. of Processors =       9      Load value       = 0.80
Total No. of jobs =  1000000     Ld Balancing Alg. = Global Avg Nbor Policy

Threshold Level  =    2.00       Distance        = 1
Convergence to < 2.00%

  TIME      VAR.     LOAD    DIFF.    RTime    %Conv    MIG.      Tx      %JOB
  ----      ----     ----    -----    -----    -----    ----      --      ----
1800.00    1.2337   2.8964  2.9672   2.2577    3.20    0.6477    8.1036   1.29
3600.00    1.2880   2.9055  3.0283   2.2175    0.88    0.6530    8.1900   2.58
3610.00    1.2867   2.9041  3.0266   2.2175    0.94    0.6529    8.1880   2.59

*** TERMINATING CONDITION : CONVERGENCE @2.00%***

SIMULATION COMPLETED ON Tue Jun  1 17:36:01 1993
------------------------------------------------
```

```
SIMULATION no.3 STARTED ON Tue Jun  1 17:36:01 1993

No. of Processors =       9      Load value      = 0.80
Total No. of jobs =  1000000     Ld Balancing Alg. = Global Avg Nbor Policy

Threshold Level  =    2.00       Distance        = 2
Convergence to < 2.00%


  TIME      VAR.    LOAD   DIFF.   RTime   %Conv   MIG.     Tx     %JOB

....................
```

What is required is to extract the convergence results for each load balancing policy, computing the duration of the simulation run and the seeding used for the random number generator. The following script command would achieve this objective:

xcel run_file [No_Alg]

The parameters required are the name of the run file and the number of load balancing policies being used. Thus, the above output would be transformed as follows:

```
3600.00    0.7792   3.9503   2.4364   3.1571    2.85    0.8968   35.8049   2.58
..........
Global Average Neighbour Policy/Distance 4


3610.00    1.2867   2.9041   3.0266   2.2175    0.94    0.6529   8.1880    2.59
........
Global Average Neighbour Policy/Distance 1

....................
.........
Global Average Neighbour Policy/Distance 2
```

The resulting file can then be ported into an spreadsheet package (such as EXCEL) where statistical analysis can then take place. The listings below are of all the scripts required to produce the above output.

RAND_AWK

```
{
 if ($2 == "erand48")
 {
     for (i= 0; i < Rmax; i++)
     {
             print $5""$6"       "$8
     }
 }
}
BEGIN {
    value1 = ""
    prev   = ""
}

{
 if ($1 == "SIMULATION")
 {
  if  ($1 == prev)
  {

print value1
             print  $4" "$5" "$6" "$7" "$8
             prev = ""
  }
  else { prev = $1; value1 = $5" "$6" "$7" "$8" "$9}
 }
}
```

RTIME_AWK

```
BEGIN {
    value1 = ""
    prev   = ""
}
{
 if ($1 == "SIMULATION")
 {
  if  ($1 == prev)
  {

             print value1
             print  $4" "$5" "$6" "$7" "$8
             prev = ""
  }
  else { prev = $1; value1 = $5" "$6" "$7" "$8" "$9}
 }
}
```

STAT_AWK

```
BEGIN {
    value1 = ""
    value2 = ""
     value3 = ""
     value4 = ""
     value5 = ""
     son   = ""
}

{
 if ($7 == "Ld")
  {
   value3 = $11 "  " $12$13$14
  }

 if ($5 == "Probe")
  {
   value5 = $8
  }

 if ($5 == "Distance")
  {
   value5 = $7
  }

 if ($2 == "TERMINATING")
  {
   son = "     " value2 "  " value3 value5 "\t"
  }
  else

 if ($1 == "Final")
  {
   value4 = substr($3,1, index($3,",") -1) "\t" $6
   print son value4
  }
  {
   value2 = value1
   value1 = $0
  }
}
```

ALG_AWK

```
BEGIN {
     value2 = $0
     AccTime  = 0
     count  = 0
     name   = ""
     prevname = ""
}

{
 if (idx % algs == 0)
 {
  print value2
  count++
  AccTime += $5
  name = prevname
 }
 idx++
 prevname = $10$11
 value2 = $1"\t"$2"\t"$3"\t"$4"\t"$5"\t"$7"\t"$8
 value2 = value2 "\t"$14"\t"$6"\t"$9"\t"$12"\t"$13"\t"$15"\t"$16"\t"$17
}

END {
 if (idx % algs == 0)
 {
  print value2
 }
 print "\n AvgRT = "AccTime" / "count " = " AccTime/count"\t"name"\n"

}
```

CALC_TIME.C

```
#include  "stdio.h"
#include  <time.h>
FILE   *fp;

main (argc, argv)
int    argc;
char   *argv[];
{
 char     start_str[25], stop_str[25];
 time_t   st, et;
 struct tm start_t, end_t;
 double   sec;
 int      get_line(), end_f;

 fp = fopen(argv[1], "r");
 while (1)
 {
  end_f = get_line(start_str);
  if (end_f == EOF)
    break;
  end_f = get_line(stop_str);
  (void) strptime (start_str, "%a %h %e %T %Y", &start_t);
  (void) strptime (stop_str, "%A %B %d %T %Y", &end_t);

  st = timelocal(&start_t);
  et = timelocal(&end_t);
  printf ("\t%4.2f\n", (et - st)/3600.0);
  /* (void) difftime(st, et); */
 }
}


int  get_line (line)
 char        *line;
{
 char  ch;
 int   i = 0;

 ch = fgetc (fp);
 while (!(ch == '\n' || ch == EOF))
 {
  i++;
  *line++ = ch;
  ch = fgetc (fp);
 }
 *line = '\0';
 if (ch == EOF)
   return EOF;
 else
   return i;
}
```

RSEED_CPL

```
#
#This script extracts the the initial seed values at the time of
#convergence from FILE1 and places them in FILE2.
#

if ($3 != "") then
    awk -f ~rsimpson/script_dir/rand_awk Rmax=$1 $2 >tr_tmp_101
    tr -d ',' < tr_tmp_101 > $3
    rm tr_tmp_101
else if ($2 != "") then
    awk -f ~rsimpson/script_dir/rand_awk Rmax=$1 $2 >tr_tmp_101
    tr -d ',' < tr_tmp_101
    rm tr_tmp_101
else
  echo "usage: rnd 0-9 run_file [seed_file]"
endif
```

CALG_CPL

```
#This script extracts the performance statistics for individual
#algorithms from the FILE1 -containing alternate data for each.
#
#The results are placed in individual temporary files before
#being concatenated and stored in FILE2.
#
#All temporary files are then erased.
#
set fid=temp
if (($1 != "") && ($2 != "")) then
 set Cx=1
 if ($3 != "")  set fid = $3
 while ($Cx <= $1)
  awk -f ~rsimpson/script_dir/alg_awk idx=$Cx algs=$1 $2 >$fid{_alg}$Cx
  @ Cx += 1
 end
 if ($3 != "") then
   cat /dev/null >$3
   foreach file  ($fid{_alg}*)
     cat $file >>$3
   end
 else
   foreach file  ($fid{_alg}*)
     cat $file
   end
   rm $fid{_alg}*
 endif
else
    echo "usage: cg 0-9 stat_file [ssort_file]"
endif
```

THIS PAGE IS BLANK

CONVERT_CPL

```
#!/bin/csh
#
# This script extracts the initial simulation trace file (FILE1)
# the results at the time of convergence from FILE1 and places them
# in FILE2.
#
if ($2 != "") then
    awk -f ~rsimpson/script_dir/stat_awk $1 >tr_tmp_101
    tr -s ' ' '\11' <tr_tmp_101 > $2
    rm tr_tmp_101
else if ($1 != "") then
    awk -f ~rsimpson/script_dir/stat_awk $1 >tr_tmp_101
    tr -s ' ' '\11' < tr_tmp_101
    rm tr_tmp_101
else
  echo "usage: cv run_file [stat_file]"
endif
```

CRSHEET_CPL

```
# This script creates the data to be used in the spreadsheet.
#
if ($2 != "") then
  cv $2  >tr_tmpst_101
  gt $2  >tr_tmpti_101
  rnd $1 $2 >tr_tmpsd_101
  paste tr_tmpst_101 tr_tmpti_101 tr_tmpsd_101 >tr_tmpRS_101
  if ($3 != "") then
     tr -s '    ' '         ' <tr_tmpRS_101 >$3
  else
     tr -s '    ' '         ' <tr_tmpRS_101
  endif
  rm tr_tmp*_101
else
  echo "usage: go 0-9 run_file [stat_file]"
endif
```

EXCEL_CPL

```
#!/bin/csh
#
#
if (($1 != "") && ($2 != "")) then
  go $2 $1 >x_tmp_101
  cg $2 x_tmp_101
  rm x_tmp_101
else
  echo "usage: xcel run_file [No_Alg]"
endif
```

GET_TIME.CPL

```
#This script will extract the start and stop times from
#each simulation run - stored in FILE1, and store them
#in FILE2.
#
set V = $2
if ($2 == "") then
  set V = temp_t123.dat
endif
if ($1 != "") then
  awk -f ~rsimpson/script_dir/rtime_awk $1 >$V
#
#The difference in time (hrs) of each run is computed
#and stored along with start and stop times in file3.
#
  if ($3 != "") then
    ~rsimpson/script_dir/calctime.exe $V >$3
  else
    ~rsimpson/script_dir/calctime.exe $V
    if ($V == "temp_t123.dat") then
      rm $V
    endif
  endif
else
  echo "usage: gt run_file [[tim_file] [hrs_file]]"
endif
```

C.3 ANSI C IMPLEMENTATION

This section presents the final and complete source code for the simulation model described in this study:

MODEL_PARAMS.H

```
#define     SUCCESS     1
#define     FAILED      0
#define     ASCEND      1
#define     DESCEND     0
#define     EMPTY       -1

#define     BOOLEAN     int

#define     TRUE        1
#define     FALSE       0

#define     QUEUES_FLUSHED  1
#define     DATA_CONVERGED  2

#define     NQUANTA         10
#define     N_SYS_PROCS     1
#define     SAMPLE_T        10
#define     SMPBLK_SZ       (SAMPLE_T + 1)
#define     TABLE_SIZE      4
#define     CSET_SIZE       8

#define     CONTEXT_SWITCH  200
#define     QUANTUM         20000
#define     ONE_SECOND      1000000.0
#define     ONE_MINUTE      (60 * ONE_SECOND)
#define     ONE_HOUR        (60 * ONE_MINUTE)
#define     BAKUP_INTERVAL  (8  * ONE_HOUR)
#define     DUMP_INTERVAL   (10 * ONE_SECOND)

#define     STABLE_STATE    3600
#define     TRACE_INTERVAL  (5 * ONE_HOUR)
#define     AVE_PROC_GROUP  1.0
#define     AVE_EXEC_TIME   1.0
#define     AVE_INST        1.0

#define             KC_CALL         0
#define             RCV_AVG_TOUT    -11
#define             EXEC_CALL       -10
#define             PRB_TOUT        -9
#define             CANCEL_TMER             -8
#define             RESET_TMER      -7
#define             SRQ_TOUT        -6
#define             WAITP_TOUT      -5
#define             LOW_TOUT        -4
#define             HIGH_TOUT       -3
#define             CREATE_CALL     -2
#define             HALT_CALL       -1

#define             KC_EMSG         1
#define             LCAL_PRB_MSG    2
```

```
#define     PSTOP_MSG       3
#define     PRB_MSG         4
#define     PRB_RPLY_MSG    5
#define     PRC_ARI_MSG     6
#define     WK_RQST_MSG     7
#define     WK_RPLY_MSG     8
#define     SND_PRC_MSG     9
#define     HIGH_LD_MSG     10
#define     NEW_LD_MSG      11
#define     LOAD_STATE      12
#define     LOW_LD_MSG      14
#define     PINFO_MSG       15

#define     TIME_EXIT       200
#define     RX_BYTE_TIME    1.0
#define     TX_BYTE_TIME    1.0
#define     PROTOCOL_TIME   1000.0
#define     TIME_SLICE      50000.0
#define     ELAPSED_TIME    10000.0
#define     SYNC_TIME       50000.0
#define     EXECUTION_TIME  (AVE_EXEC_TIME * ONE_SECOND)
#define     USERtime        1
#define     OStime          2
#define     TIME_MIG_PROC   (PROTOCOL_TIME * AVE_INST)
#define     TIME_RCV_PROC   (PROTOCOL_TIME * AVE_INST)
#define     MESS_SIZE       10000.0

#define     START_NODE          15
#define     CONVERGE_FACTOR     0.02
#define     PAGE_SIZE           500000
#define     BATCH_RUN           FALSE
/*
#define     DEBUG_DUMP          FALSE
#define     MIN_PID             800
#define     MAX_PID             1200
#define     DEBUG_MSG           TRUE
*/
#define     TIMEOUT             (EXECUTION_TIME / 5.0)
#define     GBAL_TOUT           (EXECUTION_TIME / 4.0)
#define     ARR_TIME            ( 2 * TX_BYTE_TIME + (PROTOCOL_TIME *
AVE_INST))
#define     Delay_Time     (ARR_TIME * msg->distance * 2 + TIMEOUT)
#define     context_switch time_update(node_P, (int) (CONTEXT_SWITCH * AVE_INST),
USERtime)
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  These type declaration differs from the original through the
  introduction of a buffer number and an "event filename" per cpu.

  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

typedef struct processor_node    PROCESSOR;
typedef struct process_entry     USER_PROCESS;
typedef struct comm_packet       MSG_FRAME;
typedef struct data_frame        GLBAL_FRAME;

typedef struct
     {
       int  actual_load;
       int  virtual_load;
       int  OSportion;
       int  used;
     }
     QTUM_ENTRY;

typedef struct
     {
      short int          node_id;
      PROCESSOR         *node_ptr;
     }
     ROUTE_ENTRY;

typedef struct
     {
       int                node_id,
                          hops;
     }
     LINK_ENTRY;

typedef struct
     {
      short int        id,
                       ld;
     }
     LOAD_ENTRY;

typedef struct
     {
       BOOLEAN            await_sync;
       PROCESSOR          *node_adr;
     }
     NODE_ENTRY;
```

```
struct process_entry
{
  int                 upid;
  int                 residency;
  int                 n_probes;
  double              exec_time;
  int                 exec_here_time;
  int                 residency_time;
  double              exist_time;
  int                 orig_mc,
                      preferred_mc;
  BOOLEAN             schedulable,
                      blocked,
                      finished,
                      mcs_probed[6];
  double              timeout;
  int                 itype;
  USER_PROCESS        *nxt_proc_ptr;
};

struct comm_packet
{
  double              time_stamp;
  int                 sender_id,
                      dest_id,
                      distance,
                      total_bytes,
                      type,
                      service_no;
  double              ldvalue;
  BOOLEAN             ack_reply;
  USER_PROCESS        *mess_data;
  MSG_FRAME           *next_pack_ptr;
};

struct data_frame
{
  int                 size,
                      type,
                      sno;
  USER_PROCESS        *prc;
  double              ldv;
};

typedef struct
    {
      double          mintime;
      int             itype;
      USER_PROCESS *minP;
    }
    TMER_QUEUE;

typedef struct
    {
      MSG_FRAME   *head,
                  *tail;
    }
    MESS_QUEUE;
```

```
struct processor_node
{
  QTUM_ENTRY        quanta[NQUANTA];
  int               quanta_idx,
                    OSoverhead;

  short int         node_id;
  unsigned short    xsubi[3];

  LINK_ENTRY        *links;
  LOAD_ENTRY        cSet[CSET_SIZE];

  int          load,
               threshold,
               n_tx, n_rx,
               n_migrates,
               n_immigs,
               n_deaths,
               n_local_procs,
               n_active_local_procs,
               n_virtual_procs;
  float        wt;

  BOOLEAN  high_t_set,
           low_t_set,
           pwait_set,
           rplimit;

  double       sys_real_time,
               next_elapsed_second,
               last_perf_dump_time;

  ROUTE_ENTRY         route_table[TABLE_SIZE];

  double          nxt_arr_time,
                  cumul_exist_time;

  USER_PROCESS        *curr_process,
                      *firstjb_ptr,
                      *lastjb_ptr;
  MESS_QUEUE          queue;
  TMER_QUEUE          timq;

  short int          buffer_no;
  char               event_fname[16];

  FILE               *job_file,
                     *trace_file;
};
```

```
/* ============================================================== */
/*    This version of initb_module on SparcStation One has         */
/*    been modified inorder to enable meshes greater than          */
/*    4x4 to be modelled.  This was achieved by the removal        */
/*    of a recursive descent search in building the 'shortest      */
/*    path route table.                                            */
/*                                                                 */
/*    Date of Modification: 27th March 1991                        */
/* ============================================================== */
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <math.h>
#include "model_params.h"
#include "model_types.h"
#include "globvar.h"

extern FILE      *trace,
                 *seedfile,
                 *monitor_file,
                 *configfile,
                 *paramfile;

typedef struct stack_entry
{
  short int             node_id,
                        distance;
  BOOLEAN              visited;
  ROUTE_ENTRY          aroute;
  struct stack_entry   *next;
} STACK_ENTRY;

NODE_ENTRY          *processor_table;

int            *VTable;
float           VTotal;
int             mesh_length;
extern int      total_pe;

STACK_ENTRY         *main_stak,
                    *dum_stak;

static int      gbal_stak[36],
                gbal_t;

BOOLEAN     Key_In_List (int [],int, int);
void        free_qspace(STACK_ENTRY *),
            push(STACK_ENTRY **, STACK_ENTRY *);
BOOLEAN     Balanced_Route (PROCESSOR *, int);

STACK_ENTRY  *Create_ItemQ(int, BOOLEAN,int, PROCESSOR *),
             *pop(STACK_ENTRY **);
```

```
/* ================================================================ */
/*   A v e _ C a l l                                                 */
/*                                                                   */
/* ================================================================ */
int  Ave_Call (PROCESSOR *node_P)
{
  int       idx,
            sum   = 0,
            nbors = 0;

  for (idx = 0; idx < TABLE_SIZE; idx++)
    if (node_P->route_table[idx].node_id == EMPTY)
      continue;
    else
    {
       sum += VTable[node_P->route_table[idx].node_id];
       nbors++;
    };
  return ((sum)/ nbors);
}

/* ================================================================ */
/*    d u m p _ s t a c k                                            */
/*                                                                   */
/* ================================================================ */
void    dump_stack(STACK_ENTRY *top)
{
 while (top != NULL)
 {
  printf ("\nP%d\tN%d\tkm = %d\t", top->node_id,
        top->aroute.node_id, top->distance);
  if (top->visited)
    printf ("TRUE");
  else
    printf ("FALSE");
  top = top->next;
 }
 printf ("\n\n");
}
```

```
/* ============================================================ */
/*   R e a c h a b l e                                           */
/*                                                               */
/* ============================================================ */
BOOLEAN  Reachable (int pid, int sn, int dn, int km)
{
   int      tp = 0, stak[36];
   int      i, nd, cpy;

   stak[tp++] = -(pid + 1);
   cpy = km;
   do{
      stak[tp++] = -(sn + 1);
      for (i = 0; i < TABLE_SIZE; i++)
      {
       nd  = processor_table[sn].node_adr->route_table[i].node_id;
       if (nd == EMPTY)
         continue;
       if (Key_In_List(stak, tp, nd + 1));
       else
         stak[tp++] = nd + 1;
      }
      km --;
/*
      printf ("\nReachability Stack Dump km =%d\n", km);
      for (i = 0; i < tp; i++)
        printf ("%d\n", stak[i]);
*/
      if (Key_In_List(stak, tp, dn + 1) && (km == 1))
        return TRUE;
      if (km <= 1)
      {
        km++;
        while (stak[--tp] >= 0);
      }
      if (tp > 1)
      {
        while ((tp != 1) && (stak[--tp] < 0)) km++;
        sn = stak[tp] - 1;
      }
   }while (tp != 1);
   return FALSE;
}

/* ============================================================ */
/*   K e y _ I n _ L i s t                                       */
/*                                                               */
/* ============================================================ */
BOOLEAN  Key_In_List (int stk[], int t, int key)
{
  int i;

  for (i = 0; i < t; i++)
   if ( key == (int) fabs((double)stk[i]))
     return TRUE;
  return (FALSE);
}
```

```
/* ============================================================ */
/*  U p d a t e _ L i n k                                        */
/*                                                               */
/* ============================================================ */
BOOLEAN  Update_Link (PROCESSOR *src, int nbor,STACK_ENTRY *top)
{
  int       dn;

  dn = top->aroute.node_id;
  if (dn != src->node_id)
    if (Reachable (src->node_id, nbor, dn, top->distance))
    {
     if (src->links[dn].node_id < 0)
     {
      VTable[nbor]+= 1;
      src->links[dn].node_id = nbor;
      src->links[dn].hops   = top->distance;
     }
     free_qspace(top);
     return (TRUE);
    };
  push (&main_stak, top);
  return (FALSE);
}

/* ============================================================ */
/*  w r i t e _ i n t r o _ s c r i p t                          */
/*                                                               */
/* ============================================================ */
void  write_intro_script (void)
{
  char      net_script[512];
  int       idx;

  for (idx = 0; idx < 512 ; idx++)
  {
   if (feof(configfile))
     break;
   if ((net_script[idx] = getc (configfile)) == '!')
     break;
  }
  net_script[idx] = '\0';
  idx = 0;
  while (net_script[idx] != '\0')
  {
   putc (net_script[idx], monitor_file);
   idx++;
  }
  fscanf (configfile, "%d", &total_pe);
}
```

```
/* ================================================================ */
/*     Create_proc_node                                              */
/*                                                                   */
/* ================================================================ */
int   Create_proc_node (int node_id,NODE_ENTRY processor_table[])
{ /* implementation */
  PROCESSOR    *new_node;

  new_node = (PROCESSOR *) malloc (sizeof(PROCESSOR));
  if (new_node == NULL)
    return FAILED;
  else
   { int     idx;
    /* Construct data node */
    processor_table[node_id].node_adr  = new_node;
    new_node->node_id          = node_id;
    new_node->links = (LINK_ENTRY *)
                calloc (total_pe, sizeof (LINK_ENTRY));
    for (idx = 0; idx < total_pe; idx++)
    {
      new_node->links[idx].node_id          = EMPTY;
      new_node->links[idx].hops            = total_pe;
     }
    for (idx = 0; idx < 4; idx++)
    {
      new_node->route_table[idx].node_ptr = NULL;
      new_node->route_table[idx].node_id  = EMPTY;
     }
    return SUCCESS;
   }
}
```

```
/* ============================================================== */
/*    auto_link_creation                                          */
/*                                                                */
/* ============================================================== */
void   auto_link_creation (FILE *configfile, int mesh_len)
{
  int  node_no,
       row_idx, col_idx;

  node_no = 0;
  for (row_idx = 0; row_idx < mesh_len; row_idx++)
  {
    for (col_idx = 0; col_idx < mesh_len; col_idx++)
    {
      fprintf (configfile, "%d\n", node_no);
      if (row_idx < (mesh_len -1))
      {
        fprintf (configfile, "%d  %d\n",
                 node_no, node_no + mesh_len);
        if (row_idx > 0)
          fprintf (configfile, "%d  %d\n",
                   node_no, node_no - mesh_len);
      }
      else
        fprintf (configfile, "%d  %d\n",
                 node_no, node_no - mesh_len);
      if (col_idx < (mesh_len -1))
      {
        fprintf (configfile, "%d  %d\n",
                 node_no, node_no + 1);
        if (col_idx > 0)
          fprintf (configfile, "%d  %d\n",
                   node_no, node_no - 1);
      }
      else
        fprintf (configfile, "%d  %d\n",
                 node_no, node_no - 1);
      node_no++;
    }
  }
  fprintf (configfile, "%d\n", node_no);
}
```

```
/* ============================================================ */
/*     manual_link_create                                        */
/*                                                               */
/* ============================================================ */
void   manual_link_create (FILE *configfile, int maxmcs)
{
 int    link_no,
        node_id;

 for (node_id = 0; node_id < maxmcs;)
 {
  fprintf (configfile, "%d\n", node_id);
  printf ("\nNode%d is:", node_id);
  for (;;)
  {
    printf ("connected to ->");
    scanf ("%d", &link_no);
    if ((link_no < 0) || (link_no == node_id)
       || (link_no > maxmcs))
      break;
    fprintf (configfile, "%d    %d\n", node_id, link_no);
  }
  node_id++;
 }
 fprintf (configfile, "%d\n", node_id);
}

/* ============================================================ */
/*     square_mesh                                               */
/*                                                               */
/* ============================================================ */
int    square_mesh (int total_pe)
{
 double  value;

 value = sqrt((double) total_pe);
 return (int) (floor (value));
}
```

```
/* ============================================================= */
/*    create_config_file                                         */
/*                                                               */
/* ============================================================= */
void   create_config_file (BOOLEAN batch_run)
{
 char  response[3],
       net_description[512];
 int   pe_max, prev_total = 0;

 if (!batch_run)
 {
  printf ("\nDo you wish to create a new configuration file?");
  scanf ("%s", response);
  if (strcmp(response, "no\0"))
  {
    configfile = fopen ("config.dat", "w");
    for (;;)
    {
     printf ("\n\nEnter the total number of PE's ");
     scanf ("%d", &pe_max);
     if (pe_max <= 1)
       break;
     total_pe = pe_max;
     printf ("\nPROVIDE A BRIEF DESCRIPTION OF THE NETWORK TOPOLOGY AND
THE\n");
     printf ("\nPROCESSOR ARRANGEMENT YOU INTEND TO SIMULATE...\n");
     printf ("\nTHE TEXT MUST BE TERMINATED BY AN EXCLAMATION MARK\n\n");
     {int idx;
      for ( idx = 0; idx < 512; idx++)
      {
       if ((net_description[idx] = getc(stdin)) == '!')
         break;
      }
      net_description[++idx] = '\0';
     }
     fprintf (configfile, "%s\n", net_description);
     fprintf (configfile, "%d\n", total_pe);
     mesh_length = square_mesh (total_pe);
     if (total_pe == prev_total)
     {
      printf ("\nDo you wish to use the preceding config. job stream ? ");
      scanf ("%s", response);
      fprintf (configfile, "%d\n",
            (int) ((strcmp(response, "no\0"))));
     }else prev_total = total_pe;
     fprintf (configfile, "%d\n", mesh_length);
     if (mesh_length * mesh_length == total_pe)
     {
      printf ("\nAutomatic point-to-point connection? ");
      scanf ("%s", response);
      if (strcmp(response, "no\0"))
        auto_link_creation (configfile, mesh_length);
      else  manual_link_create (configfile, total_pe);
     }else manual_link_create (configfile, total_pe);
    }
    fclose(configfile);
  }
}
```

```
}
/* ============================================================ */
/*     create_params_file                                       */
/*                                                              */
/* ============================================================ */
void   create_params_file (BOOLEAN batch_run)
{
  void  put_lba_parameters(int);
  char  response[3];
  float load_value;
  int   total_jobs,
        lba_index;

  if (!batch_run)
  {
   printf ("\nDo you wish to create a new PARAMETER file?");
   scanf ("%s", response);
   if (strcmp(response, "no\0"))
   {
     paramfile = fopen ("pramfile.dat", "w");
     for (;;)
     {
      printf ("\nEnter the LOAD value ");
      scanf ("%f", &load_value);
      if (load_value <= 0.0)
        break;
      fprintf (paramfile, "%f\n", load_value);
      printf ("\nEnter the total number of jobs:  ");
      scanf ("%d", &total_jobs);
      if (total_jobs < 1)
          break;
      fprintf (paramfile, "%d\n", total_jobs);
      printf ("\nWill this be a new job_stream? ");
      scanf ("%s", response);
      fprintf (paramfile, "%d\n",
              (int) ((strcmp(response, "no\0"))));
      printf ("\nSelect the required LBA: ");
      scanf ("%d", &lba_index);
      fprintf (paramfile, "%d\n", lba_index);
      put_lba_parameters(lba_index);
     }
    fclose(paramfile);
   }
  }
}
```

```
/* ================================================================ */
/*    put_lba_parameters                                             */
/*                                                                   */
/* ================================================================ */
void  put_lba_parameters (int lba_idx)
{ float   threshold;
  int     probe_limit;

  probe_limit = 2 * (mesh_length - 1);
  switch (lba_idx)
  {
   case 0: /* no load balancing */
          threshold = (float) 1000;
          fprintf (paramfile, "%f\n", threshold);
          break;

   case 1: /* Random          */
   case 3: /* Reverse         */
   case 4: /* Threshold Broadcast */
   case 9: /* Global Average  - Sender   */
   case 10:/* Global Average - Receiver  */
          printf ("\nEnter the THRESHOLD level: ");
          scanf ("%f", &threshold);
          fprintf (paramfile, "%f\n", threshold);
          break;

   case 5: /* Threshold Neighbour */
   case 6: /* Adaptive Threshold  */
   case 7: /* Adaptive Reverse   */
   case 8: /* Global Average Nbor */
          printf ("\nEnter the THRESHOLD level: ");
          scanf ("%f", &threshold);
          printf ("\nEnter the Maximum Distance: ");
          scanf ("%d", &probe_limit);
          fprintf (paramfile, "%f\n", threshold);
          break;

   case 2: /* Threshold Policy */
          printf ("\nEnter the THRESHOLD level: ");
          scanf ("%f", &threshold);
          fprintf (paramfile, "%f\n", threshold);
          printf ("\nEnter the PROBE LIMIT: ");
          scanf ("%d", &probe_limit);
   default:break;
  }
  fprintf (paramfile, "%d\n", probe_limit);
}
```

```
/* ============================================================ */
/*    Build_Network                                            */
/*                                                             */
/* ============================================================ */
void  Build_Network(void)
{
 int        Create_proc_node(int, NODE_ENTRY[]);
 void       Build_route_table(NODE_ENTRY []);

 int        i,idx, lidx,
           node_id, link_id, prev_id;

 PROCESSOR    *node_ptr;

 for (idx = 0; idx < total_pe; idx++)
  if (!Create_proc_node(idx, processor_table))
   {
    break;
   }
 fscanf (configfile, "%d", &node_id);
 idx = 0;

 while ((idx < total_pe) && (!feof(configfile)))
 {
   lidx = 0;
   node_ptr = processor_table[idx].node_adr;
   prev_id  = node_id;
   fscanf (configfile, "%d", &node_id);
   while (node_id == prev_id)
   {
    fscanf (configfile, "%d", &link_id);
    node_ptr->route_table[lidx].node_id = link_id;
    node_ptr->route_table[lidx].node_ptr
       = processor_table[link_id].node_adr;
    node_ptr->links[link_id].node_id = link_id;
    node_ptr->links[link_id].hops    = 1;
    lidx++;
    fscanf (configfile, "%d", &node_id);
   }
   idx++;
 }
 VTable = (int *) calloc (total_pe, sizeof (int));
 Build_route_table(processor_table);
 for (i = 0; i < total_pe; i++)
   VTotal += VTable[i];

#ifdef DEBUG_MSG
  dump_tables (processor_table);
#endif

 for (i = 0; i < total_pe; i++)
 {
  processor_table[i].node_adr->wt = VTable[i]/ VTotal;
  VTable[i] = 1;
 }
 VTotal = total_pe;
}
```

```
/* ============================================================== */
/*    Build_route_table                                          */
/*                                                               */
/* ============================================================== */
void   Build_route_table(NODE_ENTRY processor_table[])
{
  int idx;
  void traverse_net(PROCESSOR *);

     for (idx = 0; idx < total_pe; idx++)
        VTable[idx] = 1;
     idx = START_NODE;
     do
     {
      main_stak  = NULL;
      traverse_net(processor_table[idx].node_adr);
      free_qspace(main_stak);
      idx = (idx + 1) % total_pe;
     } while (idx != START_NODE);
     VTotal = total_pe;
}

/* ============================================================== */
/*    stack_links                                                */
/*                                                               */
/* ============================================================== */
BOOLEAN   stack_links (PROCESSOR *p, int km)
{
 short int      node_id, i;
 STACK_ENTRY      *stack_item;
 BOOLEAN        stacked = FALSE,
             already_stacked (int);

 stack_item = Create_ItemQ (km, TRUE, EMPTY, p);
 push (&main_stak, stack_item);
 for (i = 0; i < TABLE_SIZE; i++)
 {
  node_id = p->route_table[i].node_id;
  if ((node_id != EMPTY) && (!already_stacked(node_id)))
  {
   push (&main_stak, Create_ItemQ (km +1, FALSE, i, p));
     stacked = TRUE;
  }
 }
 return stacked;
}
```

```
/* ============================================================ */
/*  I n c o m p l e t e _ L i n k s                              */
/*                                                              */
/* ============================================================ */
BOOLEAN Incomplete_Link (LINK_ENTRY ltab[])
{
  int       i,
            count = 0;

  for (i = 0; i < total_pe; i++)
    if (ltab[i].node_id < 0)
      count++;
  if (count == 1)
    return FALSE;
  else
    return TRUE;
}

/* ============================================================ */
/*  F i n d _ L i n k s                                         */
/*                                                              */
/* ============================================================ */
BOOLEAN Find_Links (LINK_ENTRY ltab[],int nbor,int km)
{
  int       i;
  BOOLEAN   found = FALSE, stacked;

  for (i = 0; i < total_pe; i++)
    if ((ltab[i].node_id == nbor)
       && (ltab[i].hops == km))
      stacked = stack_links(processor_table[i].node_adr, km);
      found = stacked || found;
  return stacked;
}


/* ============================================================ */
/*   P u s h _ L t a b l e                                      */
/*                                                              */
/* ============================================================ */
BOOLEAN   Push_Ltable (PROCESSOR *src)
{int          success = FALSE,
              nbour_id,
              stak[4];
 short int    i, t = 0;

 STACK_ENTRY   *top;

 top = pop(&main_stak);
 while (top ->distance != 0)
 {
   push (&main_stak, top);
   for (i = 0; i < TABLE_SIZE; i++)
   {
    nbour_id = src->route_table[i].node_id;
    if (nbour_id == EMPTY)
      continue;
    else
      if (Balanced_Route (src, nbour_id))
```

```
        {
          success = TRUE;
          break;
        }
        else
          if (!Key_In_List(stak, t, nbour_id))
            stak[t++] = nbour_id;
    }
    while (t > 0)
    {
      nbour_id = stak[--t];
      top = pop(&main_stak);
      if (Update_Link(src, nbour_id, top))
      {
        success = TRUE;
        break;
      }
    }
    top = pop(&main_stak);
  }
  push (&main_stak, top);
  return success;
}


/* ============================================================ */
/*     traverse_net                                             */
/*                                                              */
/* ============================================================ */
void    traverse_net (PROCESSOR *src)
{
  int               nbour_id, i,
                    distance;
  BOOLEAN           found, stacked = FALSE,
                    Incomplete_Link (LINK_ENTRY[]);

  distance = 1;
  push (&main_stak, Create_ItemQ(0, TRUE, EMPTY, src));
  while (Incomplete_Link(src->links))
  {
    for (i = 0; i < TABLE_SIZE; i++)
    {
      nbour_id = src->route_table[i].node_id;
      if (nbour_id == EMPTY)
        continue;
      else
      {
        Find_Links (src->links, nbour_id, distance);
        Balanced_Route (src, nbour_id);
      }
    }
    if (Push_Ltable(src))
        distance++;
    else distance--;
  }
  free_qspace(pop(&main_stak));
}
```

```
/* ============================================================== */
/*    B a l a n c e d _ R o u t e                                 */
/*                                                                */
/* ============================================================== */
BOOLEAN   Balanced_Route (PROCESSOR *src, int nbor)
{
 BOOLEAN      succeed = TRUE;
 int          dn;

 STACK_ENTRY   *top;

 top = pop(&main_stak);
 dn  = top->aroute.node_id;
 if (dn == src->node_id)
   push(&main_stak, top);
 else
 {
   if (src->links[dn].node_id < 0)
   {
     if (VTable[nbor] < Ave_Call(src))
       succeed = Update_Link(src, nbor, top);
      else
      {
        push (&main_stak, top);
        succeed = FALSE;
       }
   }
    else
    {
     free_qspace(top);
     return (Balanced_Route (src, nbor));
    }
 }
return succeed;
}

/* ============================================================== */
/*     already_stacked                                            */
/*                                                                */
/* ============================================================== */
BOOLEAN   already_stacked (int id)
{
  struct stack_entry  *dumq;

  dumq = main_stak;
  while (dumq != NULL)
  {
    if (dumq->aroute.node_id == id)
      return (TRUE);
    else
      dumq = dumq->next;
  }
  return (FALSE);
}
```

```
/* ============================================================= */
/*     free_qspace                                               */
/*                                                               */
/* ============================================================= */
void  free_qspace(struct stack_entry *q_entry)
{
   struct  stack_entry  *sublist_ptr;

   while (q_entry != NULL)
   {
      sublist_ptr = q_entry->next;
      free ((char *) q_entry);
      q_entry = sublist_ptr;
   }
}

/* ============================================================= */
/*     push                                                      */
/*                                                               */
/* ============================================================= */
void  push(STACK_ENTRY **stak, STACK_ENTRY *entry)
{
   if (entry != NULL)
   {
     if (stak  == NULL)
      *stak   = entry;
     else
     {
      entry->next  = *stak;
      *stak        = entry;
     }
    }
}

/* ============================================================= */
/*     pop                                                       */
/*                                                               */
/* ============================================================= */
STACK_ENTRY  *pop(STACK_ENTRY **stak)
{
   STACK_ENTRY  *top;

   if (*stak == NULL)
     return (NULL);
   else
    {
      top          = *stak;
      *stak        = top->next;
      top->next   = NULL;
      return (top);
    }
}
```

```
/* ============================================================ */
/*    Create_ItemQ                                              */
/*                                                              */
/* ============================================================ */
struct stack_entry  *Create_ItemQ(int km,
                        BOOLEAN visited,
                        int route_idx,
                        PROCESSOR *p)
{
  struct  stack_entry  *q_entry;

  q_entry = (struct stack_entry *) malloc (sizeof(struct stack_entry));
  if (q_entry == NULL)
    return (NULL);
  else
   {
    q_entry->node_id        = p->node_id;
    q_entry->distance          = km;
    q_entry->visited  = visited;
    if (route_idx != EMPTY)
     q_entry->aroute = p->route_table[route_idx];
   else
     {
      q_entry->aroute.node_id  = p->node_id;
      q_entry->aroute.node_ptr = p;
     };
    q_entry->next = NULL;
    return (q_entry);
   }
}

/* ============================================================ */
/*    Dump_MyLink                                               */
/*                                                              */
/* ============================================================ */
int    Dump_MyLink(LINK_ENTRY ltab[], int id)
{
 int          j;

  fprintf (monitor_file, "\n\nProcessor%d PRIMARY route\n", id);
  for (j=0; j < total_pe; j++)
  {
  /*
    fprintf (monitor_file,
          "\ndest.:%d  via:node%d  distance:%d nodes",
          j, ltab[j].node_id,
          ltab[j].hops);
  */
  }
  fprintf (monitor_file,
        "\n\n***%8d ROUTES via this node  ***\n",
        VTable[id]);
  return (0);
}
```

```
/* ============================================================== */
/*     dump_tables                                                */
/*                                                                */
/*                                                                */
/* ============================================================== */
int    dump_tables(NODE_ENTRY processor_table[])
{int          i,j;
 PROCESSOR     *node_ptr;
 for (i= 0; i < total_pe; i++)
 {
  node_ptr  = processor_table[i].node_adr;
  Dump_MyLink (node_ptr->links, i);
 }
/*    printf ("\nProcessor%d SECONDARY route\n", node_ptr->node_id);
  for (j=0; j < total_pe; j++)
  {
   fprintf (monitor_file, "\ndest.:%d  via:node%d  distance:%d nodes\n",
         j, node_ptr->links[j][2].node_id,
         node_ptr->links[j][2].hops);
  }
*/
 return (0);
}

/****   only used when debugging  the routing strategy ***
 ----------------------------------------------------
 main (void)
 {
  int    msh_size;
  create_config_file (0);
  create_params_file (0);
  configfile = fopen ("config.dat", "r");
  write_intro_script();
  processor_table = (NODE_ENTRY *)
               calloc (total_pe, sizeof(NODE_ENTRY));
  fscanf (configfile, "%d", &msh_size);
  Build_Network();
  dump_tables (processor_table);
  fclose (configfile);
 }
 ----------------------------------------------------
***       THE END                    ***/
```

```
/* ============================================================ */
/*    This version of kernal_module on SparcStation One has     */
/*    been modified so that trace dumps can be performed over   */
/*    larger and longer time intervals.                         */
/*                                                              */
/*    Date of Modification: 8th April 1991                      */
/* ============================================================ */
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include "model_params.h"
#include "model_types.h"
#include "globvar.h"

/************************************************************/
/*    CONSTANT, GLOBAL, & LITERAL                           */
/*        DEFINITIONS.                                      */
/************************************************************/
/* ** ANSI restriction **
  jmp_buf            ErrLabel;
 **   **   **/

int             Sec = 0, Del = 0,
                FinL, MaxL;
int             process_count;

double          global_clock,
                global_dump_time,
                stop_time;

unsigned short      n_reached;

char            **io_buffer_blk;

int             Virtual_Sender (PROCESSOR *),
                Virtual_Receiver(PROCESSOR *);

BOOLEAN             SkipRun,
                    finish;

extern NODE_ENTRY    *processor_table;
extern  float        threshold;
extern FILE          *monitor_file;
extern float         Mprofile;

extern int          probe_limit,
                    last_t,
                    total_pe;

extern GLBAL_FRAME     *dblock;
extern  int            lba_index;
```

```
extern void     time_update(PROCESSOR *, int, int);
extern void     send_probe(PROCESSOR *, USER_PROCESS *),
                 new_load_msg (PROCESSOR  *, MSG_FRAME  *),
                 probe_reply_msg (PROCESSOR *, MSG_FRAME *),
                 work_reply_msg (PROCESSOR *, MSG_FRAME *),
                 lcal_reply_msg ( PROCESSOR *, MSG_FRAME *),
                 Gbal_reply_msg (PROCESSOR *, MSG_FRAME *),
                 transfer_msg (PROCESSOR *, struct comm_packet *),
                 process_msg (PROCESSOR *, MSG_FRAME *),
                 exit_msg (PROCESSOR *, MSG_FRAME *),
                 exit_proc(PROCESSOR *, MSG_FRAME *);

extern BOOLEAN        probe_msg (PROCESSOR *,  MSG_FRAME *),
                      work_request_msg (PROCESSOR *, MSG_FRAME *);

extern GLBAL_FRAME     *pack_data(USER_PROCESS *, int, int);

extern void             close_eventfiles(void),
                         ave_compute_or_display(BOOLEAN),
                        final_compute(void),
                         psw_backup();
extern void             Select_next_timeout(PROCESSOR *);
extern BOOLEAN          simulation_complete(void),
                         Bcast_Cset(PROCESSOR *, int, int, int),
                         int_divisible(double, float);

extern char             *Get_Time(void);
extern int               init_eventfile(PROCESSOR *, short int);
extern MSG_FRAME        *receive(PROCESSOR *);
extern MSG_FRAME        *msg_packet (PROCESSOR *, int);
extern BOOLEAN           page_faulting(PROCESSOR *);
extern void              migrate(PROCESSOR *, USER_PROCESS *, int);
extern void              broadcast(PROCESSOR *);
extern int               bcast_nbor (PROCESSOR  *, int, int);
extern void              restart_clock(PROCESSOR *, int, double, USER_PROCESS *);
extern void              Update_Cset ( PROCESSOR *, MSG_FRAME *, int);
extern int               Cset_Load_Avg (PROCESSOR   *);
```

```
/*************************************************************/
/*                                                           */
/*   S I M U L A T E _ N E T W O R K                         */
/*                                                           */
/*************************************************************/
void   simulate_network(void)
{ PROCESSOR       *node_P;
  BOOLEAN          run_kernel(PROCESSOR []);

  void             initialise_nodes(void),
                   reset_reached(void);
  int              exit_code;

  initialise_nodes ();
  reset_reached();
  fprintf(monitor_file, "\n%8s\t%6s\t%6s\t%6s\t%6s\t%7s\t%7s\t%7s\t%5s\n",
     "TIME", "VAR.", "LOAD", "DIFF.", "RTime", "%Conv", "MIG.", "Tx", "%JOB");
  fprintf(monitor_file, "\n%8s\t%7s\t%6s\t%6s\t%6s\t%6s\t%7s\t%7s\t%5s\n",
     "----", "----", "----", "-----", "-----", "-----", "----", "--", "----");

  SkipRun = FALSE;
  finish = simulation_complete();
  while (!finish)
  { int   idx;
    for (idx = 0; idx < total_pe; idx++)
    {
      if (finish = !run_kernel (processor_table[idx].node_adr))
        break;
    }
    if (global_clock < stop_time)
    {
      global_clock = global_clock + ELAPSED_TIME;
    }
    if ((n_reached >= total_pe)
      && (global_clock >= stop_time))
    {
      reset_reached();
      if (global_clock >= global_dump_time + DUMP_INTERVAL)
      {
        if (int_divisible(global_clock, TRACE_INTERVAL) == TRUE)
        {
          ave_compute_or_display(TRUE);
          fflush (monitor_file);
        }
        global_dump_time = global_clock;
      }
    }
    exit_code = simulation_complete();
    finish    = finish || exit_code || SkipRun;
```

```
/* ** ANSI restriction  ** *
   if (setjmp(ErrLabel) > 0)
   {
        fprintf (monitor_file,
         "\n\n::M A L L O C ********** FAILED");
       fprintf (monitor_file,  "in msg_packet()\n");
       fprintf (monitor_file, "Process No.%d\n", process_count);
       finish = TRUE;
   }
**   **   */
  } /* END SIMULATION */
  final_compute();
  Sec = (int) (global_clock / ONE_SECOND);
  fprintf (monitor_file, "\n *** TERMINATING CONDITION :");
  if (exit_code == DATA_CONVERGED)
    fprintf (monitor_file,
          " CONVERGENCE @%2.2f%%***",
              100 * CONVERGE_FACTOR);
  else
    fprintf (monitor_file, " JOB STREAM EXHAUSTED ***");
  fprintf (monitor_file,
          " CR/Sec: %2.2f, DEL/Sec: %2.2f\n",
           (float) (process_count/Sec),
           (float) (Del/Sec));
  fprintf (monitor_file,
          "\nFinal Th: %d, Max Th: %d\n", FinL, MaxL);
  fprintf (monitor_file, "\n\nSIMULATION COMPLETED ON %s\n", Get_Time());
  /* Dump_RTime(); */
  close_eventfiles();
} /* END simulate_network() */
```

```
/*******************************************/
/*                                         */
/*   INITIALISE_NODES                      */
/*                                         */
/*******************************************/
void    initialise_nodes (void)
{
 extern  float  Mean_load,
                sigma_load,
                sigma_var,
                sigma_rtime,
                sigma_migrate,
                sigma_Tx;

  extern int        sigma_diff;
  short int         idx;
  char              trace_fname[16];
  PROCESSOR  *node_P;

 global_clock  = global_dump_time = 0.0;
 stop_time     = 0.0;
 process_count = 0; Del = 0;
 Mean_load     = 0.0;

 sigma_load   = 0.0,  sigma_var    = 0.0,
 sigma_rtime  = 0.0,  sigma_Tx     = 0.0,
 sigma_migrate = 0.0;
 sigma_diff   = 0;
 last_t = -1;

 Mprofile     = 0.0; FinL = MaxL = (int) threshold;

 io_buffer_blk = (char **) calloc (total_pe, sizeof(char *));
 for (idx = 0; idx < total_pe; idx++)
 {
  node_P = processor_table[idx].node_adr;
  time_update (node_P, (int) CONTEXT_SWITCH, OStime);
  *(io_buffer_blk + idx) = calloc (BUFSIZ, sizeof(char));
  sprintf (node_P->event_fname, "jobs%d_pg%d",
        idx, node_P->buffer_no);
  (void) init_eventfile (node_P, idx);
  node_P->wt = GBAL_TOUT;

#ifdef DEBUG_DUMP
    sprintf (trace_fname,"trace%d_run%d",
          node_P->node_id, run_idx);
    node_P->trace_file = fopen (trace_fname, "w");
#endif
 }
}
```

```
/*************************************************************/
/*                                                           */
/*    r e s e t _ r e a c h e d                              */
/*                                                           */
/*************************************************************/
void   reset_reached(void)
{int   i;

  n_reached  = 0;
  for (i = 0; i < total_pe ; i++)
  {
     processor_table[i].await_sync = FALSE;
  }
  stop_time = stop_time + SYNC_TIME;
}

/*************************************************************/
/*                                                           */
/*    k e r n a l _ m s g _ h a n d l e r                    */
/*                                                           */
/*************************************************************/
void kernal_msg_handler (PROCESSOR *node_P,MSG_FRAME *msg)
{
 void          kernal_call(PROCESSOR *, MSG_FRAME *),
               external_message(PROCESSOR *, MSG_FRAME *);

 if (msg != NULL)
 {
     if (msg->type == KC_CALL)
     {
       kernal_call(node_P, msg);
     }
     else
      {
      external_message (node_P, msg);
      }
     msg->mess_data = NULL;
     free ((char *) msg);
  }
}
```

```
/******************************************************************/
/*                                                                */
/*   S E R V I C E _ R O U T I N E                               */
/*                                                                */
/******************************************************************/
void Service_routine (PROCESSOR *node_P)
{
 void          prb_timout_msg(PROCESSOR *, USER_PROCESS *),
               srq_timout_msg(PROCESSOR *, USER_PROCESS *),
               low_timeout_msg(PROCESSOR *);

 USER_PROCESS      *proc_P,
                   *prev_P;

 proc_P  = node_P->firstjb_ptr;
 prev_P  = proc_P;
 do
 {
  if (proc_P == NULL)
    proc_P = node_P->firstjb_ptr;
  else
   {
    if ((proc_P->timeout)
       && (proc_P->timeout <= node_P->sys_real_time))
    {
     if (proc_P->itype == SRQ_TOUT)
       srq_timout_msg (node_P, proc_P);
     else
       if (proc_P->itype == PRB_TOUT)
         prb_timout_msg(node_P, proc_P);
    }
    proc_P = proc_P->nxt_proc_ptr;
   }
   if (proc_P == prev_P)
     break;
 }while (proc_P != prev_P);
}
```

```
/*******************************************************************/
/*                                                                 */
/*   T I M E O U T _ H A N D L E R                                 */
/*                                                                 */
/*******************************************************************/
BOOLEAN TimeOut_Handler(PROCESSOR *nodeP)
{
 void     wrq_timout_msg(PROCESSOR *),
        hi_timeout_msg(PROCESSOR *),
         Avgld_tout_msg(PROCESSOR *),
        low_timeout_msg(PROCESSOR *);

 if (nodeP->timq.itype)
 {
   if (nodeP->timq.mintime <= nodeP->sys_real_time)
   {
     switch (nodeP->timq.itype)
     {
      case WAITP_TOUT: wrq_timout_msg(nodeP);
                 nodeP->rplimit = 0;
                 nodeP->pwait_set = 0;
                 nodeP->n_virtual_procs = 0;
                 break;
      case HIGH_TOUT : hi_timeout_msg(nodeP);
                 break;
      case LOW_TOUT  : low_timeout_msg(nodeP);
                 break;
      case RCV_AVG_TOUT :
                 Avgld_tout_msg(nodeP);
                 break;
          default: Service_routine(nodeP);
     }
     Select_next_timeout(nodeP);
     return TRUE;
   }
   else
     if (nodeP->timq.minP != NULL)
       Service_routine(nodeP);
   }
 return FALSE;
}
```

```
/*****************************************************************/
/*                                                               */
/*    r u n _ k e r n e l                                        */
/*                                                               */
/*****************************************************************/
BOOLEAN run_kernel(PROCESSOR *node_P)
{
 USER_PROCESS        *proc_P;

 void           kernal_msg_handler(PROCESSOR *,MSG_FRAME *),
                schedule_process(PROCESSOR *);

 int            nxt_job,
                sender_initiated(PROCESSOR *, USER_PROCESS *);
 BOOLEAN   suspend_kernal,
                retry;
 MSG_FRAME          *msg_entry;

 if (node_P->sys_real_time < stop_time)
 do
 {
  msg_entry      = receive (node_P);
  kernal_msg_handler(node_P, msg_entry);
  TimeOut_Handler(node_P);
  if (msg_entry == NULL)
    break;
 } while (1);
 suspend_kernal = (node_P->sys_real_time >= stop_time);
 if (!suspend_kernal)
 {
  if (feof(node_P->job_file))
  {
  if (!(page_faulting (node_P)))
      return FAILED;
  }
  if (!feof(node_P->job_file))
     if (node_P->sys_real_time >= node_P->nxt_arr_time)
     { /* context_switch; */
      proc_P = Add_Process(process_count, NULL, node_P);
      sender_initiated (node_P, proc_P);
      fread ((char *)&nxt_job, sizeof(nxt_job), 1, node_P->job_file);
      fread ((char *)&(node_P->nxt_arr_time), sizeof(node_P->nxt_arr_time),
           1,
           node_P->job_file);
      process_count++;
     }; /* end outermost if: next event */
```

```
        if (node_P->sys_real_time > node_P->wt)
        {
        node_P->wt = node_P->sys_real_time + GBAL_TOUT;
        switch (lba_index)
        {int     xload;
           case 2:
           case 3:
           case 4:
           case 5: dblock =  pack_data (NULL, LOAD_STATE, 4);
                 xload = node_P->n_local_procs + node_P->n_virtual_procs;
                 if (Bcast_Cset(node_P, xload, threshold, 0));
                 else
                   Bcast_Cset(node_P, xload, threshold, 1);
                 free ((char *) dblock);
                 break;
           case 6:
           case 9: Virtual_Sender (node_P);
                break;
           case 7:
           case 10:Virtual_Receiver (node_P);
                break;
           default:;
          };
        }
        schedule_process (node_P);
#ifdef DEBUG_DUMP
     if (node_P->sys_real_time
          >= (node_P->last_perf_dump_time + DUMP_INTERVAL))
     {
      performance_dump(node_P);
       node_P->last_perf_dump_time = node_P->sys_real_time;
     }
#endif
  }
  else
   {
     if (!processor_table[node_P->node_id].await_sync)
     {
       n_reached++;
       processor_table[node_P->node_id].await_sync = TRUE;
     }
   };
   return SUCCESS;
}
```

```
/****************************************************************/
/*                                                              */
/*    SENDER_INITIATED                                          */
/*                                                              */
/****************************************************************/
int sender_initiated (PROCESSOR *node_P, USER_PROCESS *proc_P)
{ int     xload,
          thresh_Bcast (PROCESSOR *, USER_PROCESS *,int),
          random_policy(PROCESSOR *, USER_PROCESS *),
          threshold_policy(PROCESSOR *, USER_PROCESS *);

  switch (lba_index)
  {
    case 8: /* Global Symmetric -Creation/Deletion */
          return Virtual_Sender (node_P);

    case 11: /* Global Symmetric -Creation/Deletion */
          return Virtual_Receiver (node_P);
    default:
     xload = node_P->n_local_procs;
     if (xload > threshold + N_SYS_PROCS)
     switch (lba_index)
     {
     case 1: /* Random lb strategy */
          return (random_policy(node_P, proc_P));

     case 2: /* Threshold lb strategy */
         return (threshold_policy(node_P, proc_P));

     case 5: /* Threshold Neighbour*/
     case 4: /* Threshold Broadcast */
          return (thresh_Bcast(node_P, proc_P, xload));
     default:;
     };
  }
  return (FAILED);
}
```

```
/*****************************************************************/
/*                                                               */
/*    VIRTUAL_SENDER                                             */
/*                                                               */
/*****************************************************************/
int Virtual_Sender (PROCESSOR *nodeP)
{
   int    xload,
          global_avg_snd(PROCESSOR *,int),
          global_avg_rcv(PROCESSOR  *),
          Rev_avg_snd(PROCESSOR *, int);

   xload = nodeP->n_local_procs + nodeP->n_virtual_procs;
   if (xload  > nodeP->threshold + N_SYS_PROCS)
   switch (lba_index)
   {
    case 6: /* Adaptive Threshold */
          return Rev_avg_snd (nodeP, xload);
    case 8: /* Global Symmetric -Creation/Deleteion */
    case 9: /* Global Time Interval */
          return global_avg_snd (nodeP, xload);
    default:;
   }
   else
    if (xload <= nodeP->threshold)
    switch (lba_index)
    {
          break;
     case 8: /* Global Symmetric */
     case 9:
         return global_avg_rcv (nodeP);
     default:;
    };
   return (FAILED);
}
```

```
/****************************************************************/
/*                                                              */
/*    V I R T U A L _ R E C E I V E R                           */
/*                                                              */
/****************************************************************/
int Virtual_Receiver (PROCESSOR *nodeP)
{ int    xload,
         Rev_avg_snd(PROCESSOR *, int),
         Rev_avg_rcv (PROCESSOR *, int);

   xload = nodeP->n_local_procs + nodeP->n_virtual_procs;
   if (xload  <= nodeP->threshold)
   switch (lba_index)
   {
    case 10: /* Global Receiver */
          return Rev_avg_rcv (nodeP, xload);
    default:;
   }
   else
    if (xload > nodeP->threshold + N_SYS_PROCS)
    switch (lba_index)
     {
    case 10: /* Global Receiver */
          Rev_avg_snd (nodeP, xload);
    default:;
     };
   return (FAILED);
}

/****************************************************************/
/*                                                              */
/*    R E C E I V E R _ I N I T I A T E D                       */
/*                                                              */
/****************************************************************/
int receiver_initiated (PROCESSOR *node_P)
{   int      xload,
             reverse_policy (PROCESSOR *, int);

  switch (lba_index)
  {
    case 8: /* Global Symmetric -Creation/Deleteion */
          return Virtual_Sender (node_P);
    case 11: /* Global Symmetric -Creation/Deleteion */
          return Virtual_Receiver (node_P);
    default:
      xload = node_P->n_local_procs + node_P->n_virtual_procs;
      if (xload <= threshold)
      switch (lba_index)
      {
        case 3: /* Reverse lb strategy */
             return (reverse_policy(node_P, xload));
        default: ; /*   Lb 0 - no load balancing
               Lb 3 - Reverse - receiver-initiated */
      }
  }
 return (FAILED);
}
```

```
/*****************************************************************/
/*                                                               */
/*    RANDOM POLICY                                              */
/*                                                               */
/*****************************************************************/
int random_policy (PROCESSOR *node_P, USER_PROCESS *proc_P)
{ int            dest_id;

  dest_id = randmc_id(node_P);
  proc_P  = Remove_Process(node_P, proc_P->upid);
  if (proc_P != NULL)
    {
     migrate (node_P,
            proc_P,
            dest_id);
     node_P->n_local_procs--;
     node_P->n_active_local_procs--;
    }
    return SUCCESS;
}

/*****************************************************************/
/*                                                               */
/*    THRESHOLD POLICY                                           */
/*                                                               */
/*****************************************************************/
int threshold_policy ( PROCESSOR *node_P,USER_PROCESS *proc_P)
{
   proc_P->schedulable = FALSE;
   node_P->n_active_local_procs--;
   send_probe(node_P, proc_P);
   return SUCCESS;
}

/*****************************************************************/
/*                                                               */
/*    THRESH _ B C A S T                                         */
/*                                                               */
/*****************************************************************/
int thresh_Bcast (PROCESSOR *nodeP, USER_PROCESS *procP,int xload)
{

    procP->schedulable = FALSE;
    nodeP->n_active_local_procs--;
    dblock =   pack_data (procP, PRB_MSG, 4);
    nodeP->rplimit++;
    if (Bcast_Cset(nodeP, xload, threshold, 0));
    else
      bcast_nbor(nodeP,
             probe_limit,
             xload);
    free ((char *)dblock);
    restart_clock(nodeP, SRQ_TOUT,
             2 * probe_limit  * ARR_TIME + TIMEOUT,
             procP);
    return SUCCESS;
}
```

```
/****************************************************************/
/*                                                              */
/*    REVERSE  POLICY                                           */
/*                                                              */
/****************************************************************/
int reverse_policy (PROCESSOR *nodeP, int xload)
{
 if (!nodeP->pwait_set)
 {
     /* broadcast to all nodes, desire to receive work */
     dblock =  pack_data (NULL, WK_RQST_MSG, 4);

      nodeP->rplimit = Bcast_Cset (nodeP, xload, threshold, 1);
      if (!nodeP->rplimit)
        nodeP->rplimit = bcast_nbor(nodeP,
                        probe_limit,
                        xload);
     free ((char *)dblock);
     restart_clock(nodeP, WAITP_TOUT,
             2 * probe_limit * ARR_TIME + TIMEOUT,
             NULL);
     nodeP->pwait_set = nodeP->rplimit;
     nodeP->low_t_set      = 0;
 }
 return SUCCESS;
}

/****************************************************************/
/*                                                              */
/*    A D A P T I V E _ R E V E R S E                           */
/*                                                              */
/****************************************************************/
int Rev_avg_rcv (PROCESSOR *nodeP, int xload)
{

   if (nodeP->low_t_set == 0)
   {
    dblock =  pack_data (NULL, LOW_LD_MSG, 4);
    nodeP->rplimit = Bcast_Cset (nodeP, xload, nodeP->threshold, 1);
    if (!nodeP->rplimit)
      nodeP->rplimit = bcast_nbor(nodeP,
                        probe_limit,
                        xload);
    free ((char *)dblock);
    /* Add low T KCALL time_out message */
    nodeP->pwait_set = nodeP->rplimit;
    nodeP->low_t_set = 1;
    restart_clock(nodeP, LOW_TOUT,
             2 * probe_limit * ARR_TIME + TIMEOUT,
             NULL);
   }
   return TRUE;
}
```

```
/*****************************************************************/
/*                                                               */
/*    A D A P T IV E _ S E N D                                   */
/*                                                               */
/*****************************************************************/
int Rev_avg_snd(PROCESSOR *nodeP, int xload)
{
     if (nodeP->high_t_set == 0)
     {
      nodeP->high_t_set = 1;
      restart_clock(nodeP, HIGH_TOUT,
              2 * probe_limit * ARR_TIME + TIMEOUT,
              NULL);
     }
     return TRUE;
}

/*****************************************************************/
/*                                                               */
/*    G L O B A L _ A V E R A G E                                */
/*                                                               */
/*****************************************************************/
int global_avg_snd(PROCESSOR *nodeP,int xload)
{
   if (nodeP->high_t_set == 0)
   {
    dblock =   pack_data (NULL, HIGH_LD_MSG, 4);
    if (Bcast_Cset(nodeP, xload, nodeP->threshold, 0));
    else
      bcast_nbor(nodeP, probe_limit, xload);
    free ((char *)dblock);
     /* Add high T KCALL time_out message */
     nodeP->high_t_set = 1;
     restart_clock(nodeP, HIGH_TOUT,
              2 * probe_limit * ARR_TIME + TIMEOUT,
              NULL);
   }
   return TRUE;
}

/*****************************************************************/
/*                                                               */
/*    G L O B A L _ A V E R A G E                                */
/*                                                               */
/*****************************************************************/
int global_avg_rcv(PROCESSOR   *nodeP)
{
     if (nodeP->low_t_set == 0)
     {
      nodeP->low_t_set = 1;
      restart_clock(nodeP, LOW_TOUT,
              2 * probe_limit * ARR_TIME + TIMEOUT,
              NULL);
     }
     return TRUE;
}
```

```
/*****************************************************************/
/*                                                               */
/*    ADD_PROCESS                                                */
/*                                                               */
/*****************************************************************/

USER_PROCESS *Add_Process (int process_id,
                 USER_PROCESS *proc_P,
                 PROCESSOR *node_P)
{
  USER_PROCESS          *pblock,
                        *create_new_process(int, PROCESSOR *);

  node_P->n_local_procs++;
  node_P->n_active_local_procs++;
  if (proc_P == NULL)
    pblock     = create_new_process (process_id, node_P);
  else
   {
    pblock              = proc_P;
    pblock->exec_here_time = -1;
   };
  if (pblock != NULL)
  {
     pblock->itype        = 0;
     pblock->timeout      = 0;
     pblock->nxt_proc_ptr  = NULL;
     if (node_P->firstjb_ptr == NULL)
     {
      node_P->firstjb_ptr  = pblock;
      node_P->lastjb_ptr   = pblock;
     }
     else
       {
        node_P->lastjb_ptr->nxt_proc_ptr = pblock;
        node_P->lastjb_ptr           = pblock;
       }
   }
   return (pblock);
}

/*****************************************************************/
/*                                                               */
/*   KREATE_FRAME                                                */
/*                                                               */
/*****************************************************************/
MSG_FRAME   *Kreate_Frame(PROCESSOR *node_P,
                 USER_PROCESS *msg,
                 int sno)
{MSG_FRAME         *buffer;

   buffer     = msg_packet (node_P, node_P->node_id);
   buffer->type          = KC_CALL;
   buffer->service_no     = sno;
   buffer->mess_data      = msg;
   buffer->total_bytes    = 0;
   return buffer;
 }
```

```
/*****************************************************************/
/*                                                               */
/*   S C H E D U L E _ P R O C E S S                             */
/*                                                               */
/*****************************************************************/
void   schedule_process (PROCESSOR  *node_P)
{ USER_PROCESS          *proc_P,
                   *curr_process,
                   *find_process(PROCESSOR *, USER_PROCESS *);
  int               reverse_policy(PROCESSOR *, int);
  void              kernal_msg_handler(PROCESSOR *, MSG_FRAME *);

  curr_process = node_P->curr_process;
  if ((curr_process != NULL)
    && (curr_process->exec_time >= EXECUTION_TIME))
  {
     /* set up parameters */
     Del++;
     kernal_msg_handler (node_P,
                  Kreate_Frame(node_P, curr_process, HALT_CALL));
     curr_process = Remove_Process(node_P, curr_process->upid);
     proc_P  = node_P->curr_process;
     curr_process->nxt_proc_ptr = NULL;
     free ((char *) curr_process);
     receiver_initiated (node_P);
  }
  else
    proc_P  = find_process (node_P, curr_process);
  node_P->curr_process = proc_P;
  if (proc_P != NULL)
  {
     kernal_msg_handler (node_P,
                  Kreate_Frame(node_P, proc_P, EXEC_CALL));
  }
  else
   {
     time_update (node_P, (int) ELAPSED_TIME, OStime);

   }
}
```

```
/****************************************************************/
/*                                                              */
/*    FIND_PROCESS                                              */
/*                                                              */
/****************************************************************/
USER_PROCESS    *find_process(PROCESSOR *node_P,
                USER_PROCESS *proc_P)
{
  USER_PROCESS        *curr_P,
                      *prev_P;

  prev_P  = curr_P = proc_P;
  do
  {
   if (curr_P == NULL)
   {
     curr_P = node_P->firstjb_ptr;
   }
   else
      curr_P = curr_P->nxt_proc_ptr;

   if (curr_P == prev_P)
     break;
  }while ((curr_P == NULL)
       || (curr_P->schedulable == FALSE)
       || (curr_P->blocked     == TRUE));
  return curr_P;
}
```

```
/*****************************************************************/
/*                                                               */
/*   R E M O V E _ P R O C E S S                                 */
/*                                                               */
/*****************************************************************/
USER_PROCESS   *Remove_Process ( PROCESSOR *node_P, int process_id)
{
  USER_PROCESS        *curr_process,
                      *prev_Pess;

  curr_process  = node_P->firstjb_ptr;
  if (curr_process  != NULL)
  {
     prev_Pess    = curr_process;
     while (curr_process != NULL)
     {
      if (curr_process->upid == process_id)
        break;
      else
        {
         prev_Pess = curr_process;
         curr_process = curr_process->nxt_proc_ptr;
        }
     }
     if (curr_process != NULL)
     {
        if (curr_process == prev_Pess)
        {
          node_P->firstjb_ptr = curr_process->nxt_proc_ptr;
          prev_Pess = NULL;
        }
        else
           prev_Pess->nxt_proc_ptr
             = curr_process->nxt_proc_ptr;
        if (node_P->lastjb_ptr == curr_process)
          node_P->lastjb_ptr = prev_Pess;
     }
  }
  if (curr_process != NULL)
  {
    if (node_P->curr_process == curr_process)
      node_P->curr_process = curr_process->nxt_proc_ptr;
    else
       if ((node_P->curr_process != NULL)
          &&(node_P->curr_process->nxt_proc_ptr == curr_process))
         node_P->curr_process = curr_process->nxt_proc_ptr;
  }
  return curr_process;
}
```

```
/****************************************************************/
/*                                                              */
/*   C R E A T E _ N E W _ P R O C E S S                        */
/*                                                              */
/****************************************************************/
USER_PROCESS *create_new_process(int process_id,PROCESSOR *node_P)
{
  USER_PROCESS     *pblock;

  pblock     = (USER_PROCESS *)
               malloc (sizeof(USER_PROCESS));
  if (pblock != NULL)
   {
     pblock->upid          = process_id;
     pblock->orig_mc       = node_P->node_id;
     pblock->exec_time     = 0;
     pblock->exec_here_time = 0;
     pblock->exist_time    = 0;
     pblock->nxt_proc_ptr  = NULL;
     pblock->schedulable   = TRUE;
     pblock->blocked       = FALSE;
     pblock->finished      = FALSE;
     pblock->n_probes      = 0;
   }
  else
   {
     fprintf (monitor_file,
      "\n\n::M A L L O C ********** FAILED");
      fprintf (monitor_file,
              "in create_new_process()\n");
     fprintf (monitor_file, "Process No.%d\n", process_count);
     exit (0);
   }
  return pblock;
}

/****************************************************************/
/*                                                              */
/*   K E R N A L _ C A L L                                      */
/*                                                              */
/****************************************************************/
void   kernal_call ( PROCESSOR *node_P, MSG_FRAME *msg)
{
 void             do_processing(PROCESSOR *);
 USER_PROCESS       *create_new_process(int, PROCESSOR *);

 switch (msg->service_no)
 {
  /* case CREATE_CALL: create_new_process();
             break; */
  case HALT_CALL:   exit_proc(node_P, msg);
             break;
  case EXEC_CALL:   do_processing(node_P);
             break;
  case RESET_TMER:
             break;
  case CANCEL_TMER:
             break;
 }}
```

```
/****************************************************************/
/*                                                              */
/*   EXTERNAL_MESSAGE                                           */
/*                                                              */
/****************************************************************/
void   external_message (PROCESSOR *node_P, MSG_FRAME *msg)
{
 switch (msg->service_no)
 {case LOAD_STATE:
        switch (lba_index)
        {case 2:
         case 4:
         case 5: Update_Cset (node_P, msg, ASCEND);
                break;
          case 3: Update_Cset (node_P, msg, DESCEND);
        }
        break;
  case  PSTOP_MSG:   exit_msg(node_P, msg);
                break;
  case  HIGH_LD_MSG: threshold = node_P->threshold;
  case  PRB_MSG:     if (probe_msg (node_P, msg))
                  Update_Cset(node_P, msg, DESCEND);
               node_P->low_t_set = 0;
               break;

 case  PRB_RPLY_MSG:
       Update_Cset (node_P, msg, ASCEND);
       switch (lba_index)
       {
         case 2: probe_reply_msg (node_P, msg);
               break;
         case 4:
         case 5: lcal_reply_msg (node_P, msg);
               break;
         case 6:
         case 8:
         case 9:Gbal_reply_msg (node_P, msg);
       }
       break;
case  WK_RPLY_MSG:
       Update_Cset (node_P, msg, DESCEND);
       switch (lba_index)
       {
         case 3:if (node_P->pwait_set)
                work_reply_msg (node_P, msg);
              break;
         case 7:
         case 8:
         case 9:
         case 10:Gbal_reply_msg (node_P, msg);
       }
       break;
 case  PRC_ARI_MSG:  process_msg (node_P, msg);
               break;
```

```
   case LOW_LD_MSG:
   case WK_RQST_MSG: if (work_request_msg (node_P, msg))
                Update_Cset (node_P, msg, ASCEND);
              if (node_P->high_t_set)
                Cancel_Timer(HIGH_TOUT, node_P, NULL);
              break;

   case SND_PRC_MSG: transfer_msg (node_P, msg);
              break;

   case NEW_LD_MSG:  new_load_msg (node_P, msg);
              break;
  }
}

/****************************************************************/
/*                                                              */
/*   D O _ P R O C E S S I N G                                  */
/*                                                              */
/****************************************************************/
void   do_processing (PROCESSOR *node_P)
{
 context_switch;
 time_update(node_P,
        (int) (CONTEXT_SWITCH * AVE_INST + TIME_SLICE * AVE_INST),
         USERtime);
}

/****************************************************************/
/*                                                              */
/*   H I _ T I M E O U T _ M S G                                */
/*                                                              */
/****************************************************************/
void hi_timeout_msg(PROCESSOR *nodeP)
{
  int          xload;

  xload = nodeP->n_local_procs + nodeP->n_virtual_procs
       - nodeP->threshold;
  if (nodeP->high_t_set && (xload > 0))
  {
    nodeP->threshold++;
    if (lba_index >7)
    {
     xload =  nodeP->n_local_procs + nodeP->n_virtual_procs;
     dblock     =  pack_data (NULL, NEW_LD_MSG, 4);
     dblock->ldv =  (double) nodeP->threshold;
     if (Bcast_Cset(nodeP, xload, nodeP->threshold - 1, 0));
     else
        bcast_nbor(nodeP, probe_limit, xload);
     free ((char *)dblock);
    }
  }
  nodeP->high_t_set = 0;
#ifdef DEBUG_MSG
  if ((process_count >= MIN_PID)
      && (process_count <= MAX_PID))
  fprintf (monitor_file,
       "\n HIGHLD TIMED OUT for Node%d at %8.3f\tNewTh=%3d\n",
```

```
        nodeP->node_id,
        nodeP->sys_real_time/ONE_SECOND,
        (int) nodeP->threshold);
#endif
}


/**************************************************************/
/*                                                            */
/*   L O W _ T I M E O U T _ M S G                            */
/*                                                            */
/**************************************************************/
void low_timeout_msg(PROCESSOR *nodeP)
{int        xload;

  xload =  nodeP->n_local_procs + nodeP->n_virtual_procs
        - nodeP->threshold;
  if (nodeP->low_t_set && (xload < 0))
  {
   if (nodeP->threshold > 1)
   {
     nodeP->threshold --;
     if (lba_index > 7)
     {
      xload =  nodeP->n_local_procs + nodeP->n_virtual_procs;
      dblock =  pack_data (NULL, NEW_LD_MSG, 4);
      dblock->ldv = (double)nodeP->threshold;
      if (Bcast_Cset(nodeP, xload, nodeP->threshold - 1, 1));
      else
        bcast_nbor(nodeP, probe_limit, xload);
      free ((char *)dblock);
     }
   }
  }
  nodeP->low_t_set = 0;
#ifdef DEBUG_MSG
  if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
  fprintf (monitor_file,
        "\n LOWLD TIMED OUT for Node%d at %8.3f\t NewTh=%3d\n",
        nodeP->node_id,
        nodeP->sys_real_time/ONE_SECOND,
        (int)nodeP->threshold);
#endif
}
```

```
/*************************************************************/
/*                                                           */
/*    A V G L D _ T O U T _ M S G                            */
/*                                                           */
/*************************************************************/
void Avgld_tout_msg(PROCESSOR *nodeP)
{
  int       xload;

  xload =  nodeP->n_local_procs + nodeP->n_virtual_procs
         - nodeP->threshold;
  if (xload != 0)
  {
     dblock =  pack_data (NULL, NEW_LD_MSG, 4);
     nodeP->threshold = Cset_Load_Avg(nodeP);
     dblock->ldv = (double) nodeP->threshold;
     bcast_nbor(nodeP, probe_limit, nodeP->n_local_procs);
     free ((char *)dblock);
  }
  nodeP->low_t_set  = 0;
  nodeP->high_t_set = 0;
#ifdef DEBUG_MSG
  if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
  fprintf (monitor_file,
        "\n RCV AVGLD TIMED OUT for Node%d at %8.3f\t NewTh=%3d\n",
        nodeP->node_id,
        nodeP->sys_real_time/ONE_SECOND,
        (int)nodeP->threshold);
#endif
}

/*************************************************************/
/*                                                           */
/*    W R Q _ T I M O U T _ M S G                            */
/*                                                           */
/*************************************************************/
void wrq_timout_msg (PROCESSOR *node_ptr)
{
#ifdef DEBUG_MSG
  if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
  fprintf (monitor_file,
        "\n WAITP TIMED OUT for Node%d at %8.3f\n",
        node_ptr->node_id,
        node_ptr->sys_real_time/ONE_SECOND);
#endif
  /* node_ptr->pwait_set++;
  node_ptr->n_virtual_procs--;
  */
}
```

```
/**************************************************************/
/*                                                            */
/*    P R B _ T I M O U T _ M S G                             */
/*                                                            */
/**************************************************************/
void prb_timout_msg (PROCESSOR *node_ptr, USER_PROCESS *prc)
{
 /* NO REPLY RECEIVED */


  node_ptr->rplimit--;

#ifdef DEBUG_MSG
   if ((process_count >= MIN_PID)
        && (process_count <= MAX_PID))
   printf ("\n PROBE TIMED OUT for prc%d \n", prc->upid);
#endif
  node_ptr->rplimit--;
  send_probe(node_ptr, prc);
}

/**************************************************************/
/*                                                            */
/*    S R Q _ T I M O U T _ M S G                             */
/*                                                            */
/**************************************************************/
void srq_timout_msg (PROCESSOR  *node_ptr,USER_PROCESS  *prc)
{
/*  -- unblock migratable process
   -- reset node_P->SRQ_TIMOUT
*/
#ifdef DEBUG_MSG
  if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
  fprintf (monitor_file,
         "\n SRQ TIMED OUT for Node%d at %8.3f\n",
        node_ptr->node_id,
        node_ptr->sys_real_time/ONE_SECOND);
#endif

   prc->schedulable = TRUE;
   prc->itype      = 0;
   prc->timeout    = 0;
   node_ptr->n_active_local_procs++;
   if (node_ptr->rplimit > 0)
     node_ptr->rplimit--;
}
```

```
/**************************************************************/
/*                                                            */
/*    CANCEL_TIMER                                            */
/*                                                            */
/**************************************************************/
BOOLEAN Cancel_Timer (int itype,PROCESSOR *nodeP,USER_PROCESS *prc)
{
 USER_PROCESS *this_proc;

 if (prc == NULL)
 {
   if (itype == nodeP->timq.itype)
   {
    /* re-initialise */
     switch (nodeP->timq.itype)
     {
      case WAITP_TOUT: nodeP->rplimit = 0;
                  nodeP->pwait_set = 0;
                  nodeP->n_virtual_procs = 0;
                  break;
      case HIGH_TOUT : nodeP->high_t_set  = 0;
                  break;
      case LOW_TOUT  :
                  nodeP->low_t_set  = 0;
                  break;
      case RCV_AVG_TOUT:
                  nodeP->high_t_set  = 0;
                  nodeP->low_t_set  = 0;
                  break;
     }
    Select_next_timeout(nodeP);
    return TRUE;
   }
 }
 else
  {
   this_proc      = nodeP->firstjb_ptr;
   while (this_proc != NULL)
   {
    if (prc == this_proc)
    {
      if (prc->itype == itype)
      {
        /* re-initialise */
        prc->timeout = 0;
        prc->itype  = 0;
        if (nodeP->timq.minP == prc)
        Select_next_timeout(nodeP);
        return TRUE;
      }
      break;
    }
    else
      this_proc = this_proc->nxt_proc_ptr;
   }
  }
return FALSE;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <strings.h>
#include "model_params.h"
#include "model_types.h"
#include "globvar.h"

#define SYNTH_WORKLOAD      2

GLBAL_FRAME         *dblock;
MSG_FRAME           *create_frame(PROCESSOR *, int);

ROUTE_ENTRY route_node (short int, LINK_ENTRY [], ROUTE_ENTRY []);

extern int       SkipRun;
extern NODE_ENTRY    *processor_table;
extern int          probe_limit,
                    process_count,
                    total_pe;

extern  int         FinL, MaxL;


extern  float       threshold;
extern FILE         *monitor_file;


/***************************************************************/
/*                                                             */
/*   T R A N S M I T                                           */
/*                                                             */
/***************************************************************/

void transmit (PROCESSOR *sender,
          ROUTE_ENTRY nxt_node,
          MSG_FRAME *msg_P)
{
  sender->n_tx++;
  msg_P->time_stamp += PROTOCOL_TIME * AVE_INST
         + msg_P->total_bytes * TX_BYTE_TIME;
  Write_To_MsgQ(&(nxt_node.node_ptr->queue), msg_P);
  time_update (sender, (int) (PROTOCOL_TIME * AVE_INST
          + msg_P->total_bytes * TX_BYTE_TIME), OStime);
}
```

```
/*****************************************************************/
/*                                                               */
/*    R E C E I V E                                              */
/*                                                               */
/*****************************************************************/
MSG_FRAME *receive ( PROCESSOR  *node_P)
{

  ROUTE_ENTRY         next_route;
  MSG_FRAME           *msg_P;
  double          most_recent_time;

  most_recent_time = node_P->sys_real_time;
  while ((node_P->queue.head != NULL) &&
              (most_recent_time >= node_P->queue.head->time_stamp))
  {
        msg_P    = Read_From_MsgQ(&(node_P->queue));
        node_P->n_rx++;

         time_update (node_P, (int) (PROTOCOL_TIME * AVE_INST
               + msg_P->total_bytes * RX_BYTE_TIME), OStime);
         if (msg_P->dest_id == node_P->node_id)
      {
#ifdef DEBUG_MSG
   if ((process_count >= MIN_PID)
      && (process_count <= MAX_PID))
     fprintf(monitor_file, "\nNode%d receives Packet N%d-N%d::\t",
        node_P->node_id, msg_P->sender_id, msg_P->dest_id);
#endif
              return (msg_P);
      }
         else
         {
          msg_P->distance++;
          next_route  = route_node (msg_P->dest_id,
                      node_P->links,
                      node_P->route_table);
          transmit (node_P, next_route, msg_P);
         }
  }/* end while */
  return (NULL);
}
```

```
/*************************************************************/
/*                                                           */
/*   CREATE_FRAME                                            */
/*                                                           */
/*************************************************************/
MSG_FRAME   *create_frame(PROCESSOR *node_P, int dest_id)
{
 MSG_FRAME      *buffer;

   if ((buffer    = msg_packet (node_P, dest_id)) != NULL)
   {
    buffer->type          = dblock->type;
    buffer->service_no     = dblock->sno;
    buffer->mess_data      = dblock->prc;
    buffer->ldvalue       = dblock->ldv;
    buffer->total_bytes    = dblock->size;
   }
   return buffer;
}

/*---------------------------------------------------------*/
/*                                                         */
/*   msg_packet                                            */
/*                                                         */
/*---------------------------------------------------------*/
MSG_FRAME   *msg_packet (PROCESSOR *node_P, int dest_id)
{
 MSG_FRAME    *buffer;

 buffer     = (MSG_FRAME *)
              malloc (sizeof(MSG_FRAME));
 if (buffer != NULL)
 {
   buffer->dest_id       = dest_id;
   buffer->sender_id      = node_P->node_id;
   buffer->time_stamp     = node_P->sys_real_time;
   buffer->distance      = 1;
   buffer->total_bytes    = 0;
   buffer->next_pack_ptr  = NULL;
   return (buffer);
 }
 else
  { /* ** ANSI Restriction  ** *
      longjmp (ErrLabel, 3);
    **  **      */
  }
}
```

```
/****************************************************************/
/*                                                              */
/*   pack_ data                                                 */
/*                                                              */
/****************************************************************/
GLBAL_FRAME  *pack_data (USER_PROCESS *proc, int service, int sz)
{
 GLBAL_FRAME    *buffer;

 buffer     = (GLBAL_FRAME *)
              malloc (sizeof(GLBAL_FRAME));
 if (buffer != NULL)
 {
   buffer->type        = KC_EMSG;
   buffer->sno         = service;
   buffer->prc         = proc;
   buffer->size        = sz;
 }
 else
  {
     fprintf (monitor_file,
       "\n\n::M A L L O C ********** FAILED");
     fprintf (monitor_file,
              "in pack_data()\n");
     fprintf (monitor_file, "Process No.%d\n", process_count);
     exit (2);
  }
 return (buffer);
}


/****************************************************************/
/*                                                              */
/*   MIGRATABLE_TASK                                            */
/*                                                              */
/****************************************************************/
USER_PROCESS    *migratable_task( PROCESSOR *nodeP)
{
  USER_PROCESS       *proc;
  BOOLEAN            unsuitable;

  proc      = nodeP->firstjb_ptr;
  unsuitable = (proc == NULL) || (proc->schedulable == FALSE)
           || (proc->blocked == TRUE)
           || (proc == nodeP->curr_process)
           ||  (proc->exec_here_time < 0);
  do
  {
   if ((proc == NULL) || (!unsuitable))
     break;
   proc = proc->nxt_proc_ptr;
   unsuitable = (proc == NULL) || (proc->schedulable == FALSE)
          || (proc->blocked == TRUE)
          || (proc == nodeP->curr_process)
          || (proc->exec_here_time < 0);
  } while (unsuitable);
  return proc;
}
```

```
/*************************************************************/
/*                                                           */
/*    L O N G E S T _ R U N N E R                            */
/*                                                           */
/*************************************************************/
USER_PROCESS    *longest_runner(PROCESSOR *nodeP)
{
  USER_PROCESS        *proc, *lrp;
  BOOLEAN             unsuitable;

  proc      = nodeP->firstjb_ptr;
  lrp       = NULL;
  unsuitable = (proc == NULL) || (proc->schedulable == FALSE)
            || (proc->blocked == TRUE)
            || (proc == nodeP->curr_process)
            || (proc->exec_here_time < 0);
  do
  {
   if (proc == NULL)
      break;
   if (!unsuitable)
         if ((lrp == NULL)
        || (proc->exec_time < lrp->exec_time))
        lrp = proc;
   proc = proc->nxt_proc_ptr;
   unsuitable = (proc == NULL) || (proc->schedulable == FALSE)
           || (proc->blocked == TRUE)
           || (proc == nodeP->curr_process)
           || (proc->exec_here_time < 0);
  } while (1);
  return lrp;
}
```

```
/**************************************************************/
/*                                                            */
/*   R O U T E _ S E L E C T I O N                            */
/*                                                            */
/**************************************************************/
 ROUTE_ENTRY  Route_Selection(int dn,
                      LINK_ENTRY links[],
                      ROUTE_ENTRY rtable[])
{
 int        nbor_id, pe,
            km, ridx, i;

  nbor_id = links[dn].node_id;
  km      = links[dn].hops;
  for (i = 0; i < TABLE_SIZE; i++)
  {
    pe = rtable[i].node_id;
    if (pe == EMPTY)
      continue;
    else
     if (pe == nbor_id)
       ridx = i;
     else
     if (processor_table[pe].node_adr->links[dn].hops
         == km - 1)
     {
       ridx = i;
       break;
     }
   }
   links[dn].node_id = rtable[ridx].node_id;
   return rtable[ridx];
 }


/**************************************************************/
/*                                                            */
/*   R O U T E _ N O D E                                      */
/*                                                            */
/**************************************************************/
ROUTE_ENTRY  route_node (short int dest_node,
                  LINK_ENTRY links[],
                  ROUTE_ENTRY rtable[])
{
 int        idx;
 LINK_ENTRY  actual_cell;

 actual_cell = links[dest_node];
 for (idx = 0; idx < TABLE_SIZE ; idx++)
 {
    if (rtable[idx].node_id != EMPTY)
      if (actual_cell.node_id == rtable[idx].node_id)
      {
        break;
      }
 }
 return rtable[idx];
}
```

```
/*************************************************************/
/*                                                           */
/*    FIND_NBOUR                                             */
/*                                                           */
/*************************************************************/
int find_nbour (PROCESSOR    *pe, int maxd)
{
 static int       i = 0;
 auto int         j;

 if (i == total_pe)
   i = 0;
 j = i;
 do
 {
    if (pe->links[i].hops == maxd)
    {
      j = i;
      i++;
      return j;
    }
    else i = (i + 1) % total_pe;
 } while (j != i);
 return -1;
}

/*************************************************************/
/*                                                           */
/*    MIGRATE                                                */
/*                                                           */
/*************************************************************/
void    migrate(PROCESSOR *node_P,
            USER_PROCESS *proc_P,
            int dest_id)
{ ROUTE_ENTRY           next_route;

   proc_P->itype        = 0;
   proc_P->timeout       = 0;
   next_route           = route_node ((short)dest_id,
                                  node_P->links,
                                  node_P->route_table);
  if (proc_P == NULL) return;
  time_update (node_P, (int) TIME_MIG_PROC, OStime);
#ifdef DEBUG_MSG
  if ((process_count >= MIN_PID)
      && (process_count <= MAX_PID))
     fprintf (monitor_file,
          "\nnode %d sends process%d [load = %d] to node %d\n",
          node_P->node_id,
          proc_P->upid,
          node_P->load, dest_id);
#endif
  dblock =  pack_data (proc_P, PRC_ARI_MSG, 10218);
  node_P->n_migrates++;
  transmit (node_P,
          next_route,
          create_frame (node_P, dest_id));
  free ((char *)dblock);
}
```

```
/*****************************************************************/
/*    P R O B E D                                                */
/*****************************************************************/
BOOLEAN   probed (USER_PROCESS *proc_P, int mc_id)
{
 BOOLEAN found;

 found = FAILED;
 if (proc_P->n_probes > 0)
 {
  int  idx;

  idx= 0;
  while (idx <= probe_limit -1)
  {
   if ((proc_P->mcs_probed)[idx] == mc_id)
   {
     found = SUCCESS;
     break;
   };
   idx++;
  }
 };
 return found;
}
```

```
/*************************************************************/
/*                                                           */
/*    S E N D _ P R O B E                                    */
/*                                                           */
/*************************************************************/
void   send_probe (PROCESSOR *node_P, USER_PROCESS *proc_P)
{
 if (proc_P->n_probes < probe_limit)
 {
  int  mc_id,
       idx;

  idx = proc_P->n_probes;

  if ((node_P->cSet[0].id < 0)
     || (node_P->cSet[0].Id > 0))
  {
   do
   {
     mc_id = randmc_id(node_P);
   } while ((probed(proc_P, mc_id))
          || (mc_id == node_P->node_id));
  }
  else mc_id = node_P->cSet[0].id;

  proc_P->mcs_probed[idx] = mc_id;
  proc_P->n_probes++;

  /* send probe message */
  {
   int   dstance, load;
   MSG_FRAME *buff;

   dblock =  pack_data (proc_P, PRB_MSG, 4);
   buff  =  create_frame (node_P, mc_id);
   load  = (int)(node_P->n_local_procs - N_SYS_PROCS - threshold);
   buff->ack_reply = load;
   transmit (node_P,
          route_node (mc_id, node_P->links,
                      node_P->route_table),
          create_frame (node_P, mc_id));
   free ((char *)dblock);

   dstance = node_P->links[mc_id].hops;
   restart_clock (node_P, PRB_TOUT, (double)
                      ARR_TIME *  dstance * 2 + TIMEOUT,
              proc_P);
   node_P->rplimit++;

  }
 }
 else /* probe_limit has been exceeded */
  {
   node_P->n_active_local_procs++;
   proc_P->schedulable = TRUE;
  }
}
```

```
/*****************************************************************/
/*                                                               */
/*   B R O A D C A S T                                           */
/*                                                               */
/*****************************************************************/
void   broadcast (PROCESSOR *node_P)
{

 int              i;

 for (i = 0; i < total_pe; i++)
 {
  if (node_P->node_id == i)
    continue;
  else
  {
   /* send message */
    transmit (node_P,
           route_node (i, node_P->links, node_P->route_table),
           create_frame (node_P, i));
  }
 }
}

BOOLEAN Bcast_Cset(PROCESSOR *pe, int load, int threshold, int thold)
 {
  MSG_FRAME *buff;
  int i = 0, send, tx = 0;

   while ((i < CSET_SIZE) && (pe->cSet[i].id > 0))
   {
     if (thold == 0)
        send = pe->cSet[i].ld < threshold + N_SYS_PROCS;
     else
        send = pe->cSet[i].ld > threshold + N_SYS_PROCS;
     if (send)
     {
       /* send probe */
       tx++;
       buff = create_frame (pe, pe->cSet[i].id);
       buff->ack_reply = load;
       transmit (pe,
             route_node (pe->cSet[i].id,
                    pe->links,
                    pe->route_table),
             buff);
     }
     i++;
    }
   return tx;
 }
```

```
/****************************************************************/
/*                                                              */
/*    BCAST_NBOR                                                */
/*                                                              */
/****************************************************************/
int  bcast_nbor (PROCESSOR  *pe, int max_d, int load)
{
 int       i, d;
 int       total = 0;
 MSG_FRAME  *buff;

 for (i = 0; i < total_pe; i++)
 {
  d = pe->links[i].hops;
  if ((d > 0) && (d <= max_d))
   {
    /* send probe message  */
      total++;
      buff = create_frame (pe, i);
      buff->ack_reply = load;
      transmit (pe,
              route_node (i, pe->links, pe->route_table),
              buff);
   };
 }
 return total;
}



/****************************************************************/
/*                                                              */
/*    NEW_LOAD_MSG                                              */
/*                                                              */
/****************************************************************/
void new_load_msg (PROCESSOR  *nodeP, MSG_FRAME  *msg)
{
  nodeP->low_t_set  = 0;
  nodeP->high_t_set = 0;
  Cancel_Timer(LOW_TOUT, nodeP, NULL);
  Cancel_Timer(HIGH_TOUT, nodeP, NULL);
  if (msg->ldvalue > MaxL)
    MaxL = (int) msg->ldvalue;
  FinL = (int) msg->ldvalue;
  nodeP->threshold  = (int) msg->ldvalue;
}
```

```
/*****************************************************************/
/*                                                               */
/*    P R O B E _ M S G                                          */
/*                                                               */
/*****************************************************************/
BOOLEAN   probe_msg (PROCESSOR *node_P,  MSG_FRAME *msg)
{
 MSG_FRAME          *buff;
 int                xload;
 ROUTE_ENTRY         next_route;

 extern int          MsgQ_Length(),
                ProcQ_Length();

 /* received by CPU(i)  */
   dblock =  pack_data (msg->mess_data, PRB_RPLY_MSG, 5);
   buff     = create_frame (node_P, msg->sender_id);

#ifdef DEBUG_MSG
   if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
   {
    fprintf (monitor_file,
          "Node%d -overloaded- has asked node%d TO accept\n",
          msg->sender_id,
          node_P->node_id);

    fprintf (monitor_file, "msqQL=%d    procQL = %d\n",
        MsgQ_Length(node_P->queue.head),
        ProcQ_Length(node_P->firstjb_ptr));
    Dump_Msg(msg);
   }
#endif

   xload  = node_P->n_local_procs + node_P->n_virtual_procs;
   buff->ack_reply = xload;
   if (buff->ack_reply <= threshold)
   {  /* reply I am prepared to accept */
      buff->ack_reply = -(buff->ack_reply);
          node_P->n_virtual_procs++;
      node_P->rplimit++;

          restart_clock(node_P, WAITP_TOUT,
                        Delay_Time, NULL);

   }
   next_route    = route_node (msg->sender_id,
                              node_P->links,
                              node_P->route_table);
   transmit (node_P, next_route, buff);
   free ((char *)dblock);
   return xload <= threshold;
}
```

```
/************************************************************/
/*                                                          */
/*    W O R K _ R E Q U E S T _ M S G                       */
/*                                                          */
/************************************************************/
BOOLEAN   work_request_msg (PROCESSOR *node_P,MSG_FRAME *msg)
{
 MSG_FRAME          *buffer;
 USER_PROCESS        *prc,
                 *migratable_task(PROCESSOR *);
 int              xload;

 ROUTE_ENTRY          next_route;

 /* -- find a process that has not commenced execution
   -- find a process that is blocked
   -- or find a process to pre-empt
   -- Block it
 */
  prc       = NULL;
  dblock    =  pack_data (msg->mess_data, WK_RPLY_MSG, 5);
  buffer    =  create_frame (node_P, msg->sender_id);
  xload     =  node_P->n_local_procs + node_P->n_virtual_procs;
#ifdef DEBUG_MSG
  if ((process_count >= MIN_PID)
      && (process_count <= MAX_PID))
  {
   fprintf (monitor_file,
         "Node%d has asked node%d for WORK::::",
         msg->sender_id, node_P->node_id);

   fprintf (monitor_file, "\nmsqQL=%d    procQL = %d\n",
        MsgQ_Length(node_P->queue.head),
        ProcQ_Length(node_P->firstjb_ptr));
  }
#endif
  buffer->ack_reply = xload;
  if (xload > node_P->threshold + N_SYS_PROCS)
  {
     prc = longest_runner (node_P);
          if (prc != NULL)
      { /* reply - yes I have processes I wish to migrate
         buffer->mess_data  =  prc;
         prc->schedulable   =  FALSE;
         node_P->n_active_local_procs--;
         restart_clock(node_P, SRQ_TOUT, Delay_Time, prc);
         node_P->rplimit++;
         */
      } else buffer->ack_reply = -(buffer->ack_reply);
  } else buffer->ack_reply = -(buffer->ack_reply);
  next_route    = route_node (msg->sender_id,
                       node_P->links,
                       node_P->route_table);
  transmit (node_P, next_route, buffer);
  free ((char *)dblock);
  return xload > node_P->threshold + N_SYS_PROCS;
}
```

```
/****************************************************************/
/*                                                              */
/*    PROBE_REPLY_MSG                                           */
/*                                                              */
/****************************************************************/
void   probe_reply_msg (PROCESSOR *node_P, MSG_FRAME *msg_P)
{
 USER_PROCESS        *proc_P;

 /* received by CPU(i)  */
#ifdef DEBUG_MSG
   if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
     fprintf (monitor_file,
           "Node%d::Time:%f\tNode%d's reply (prc%d) was  %d\n",
           node_P->node_id,
           node_P->sys_real_time / ONE_SECOND,
           msg_P->sender_id, msg_P->mess_data->upid,
           msg_P->ack_reply);
#endif

 if (Cancel_Timer(PRB_TOUT,node_P, msg_P->mess_data))
   {
     node_P->rplimit--;

     if (msg_P->ack_reply > FALSE)
     {
      send_probe (node_P, msg_P->mess_data);
     }
     else
     {
      int  idx;
      proc_P = msg_P->mess_data;
      proc_P = Remove_Process (node_P, proc_P->upid);
      for (idx = 0; idx < proc_P->n_probes; idx++)
        proc_P->mcs_probed[idx] = EMPTY;
      proc_P->n_probes     = 0;
      migrate (node_P, proc_P, msg_P->sender_id);
      node_P->n_local_procs--;
     }
   }

}
```

```
/***************************************************************/
/*                                                             */
/*    W O R K _ R E P L Y _ M S G                              */
/*                                                             */
/***************************************************************/
void   work_reply_msg (PROCESSOR *node_P, MSG_FRAME *msg)
{
 MSG_FRAME          *buffer;
 ROUTE_ENTRY          next_route;
 int             xload;

#ifdef DEBUG_MSG
   if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
     fprintf (monitor_file,
           "Node%d::Time:%f\tNode%d's reply was  %d\n",
           node_P->node_id,
           node_P->sys_real_time / ONE_SECOND,
           msg->sender_id,
           msg->ack_reply);
#endif
   if (msg->ack_reply > FALSE)
   {

     dblock =  pack_data (msg->mess_data, SND_PRC_MSG, 5);
     buffer = create_frame (node_P, msg->sender_id);
     xload = node_P->n_local_procs + node_P->n_virtual_procs;
     buffer->ack_reply = xload;
     if (xload <= threshold)
     {
      buffer->ack_reply = -(buffer->ack_reply);
      node_P->n_virtual_procs++;
/*
          if (node_P->timq.mintime > node_P->sys_real_time + Delay_Time)
            node_P->timq.mintime = node_P->sys_real_time + Delay_Time;
*/
          restart_clock(node_P, WAITP_TOUT,
                            Delay_Time,
                  NULL);

     }
     node_P->pwait_set--;
     next_route     = route_node (msg->sender_id,
                           node_P->links,
                           node_P->route_table);
     transmit (node_P, next_route, buffer);
    }
}
```

```
/******************************************************************/
/*                                                                */
/*   LCAL_REPLY_MSG                                               */
/*                                                                */
/******************************************************************/
void  lcal_reply_msg ( PROCESSOR *node_P, MSG_FRAME *msg_P)
{
 USER_PROCESS        *proc_P;

 /* received by CPU(i)  */
 if (node_P->rplimit)
 {
  node_P->rplimit--;
  if (msg_P->ack_reply <= FALSE)
  {
    proc_P = msg_P->mess_data;
   /* Cancel Timer */
   if (Cancel_Timer(SRQ_TOUT, node_P, proc_P))
   {
     node_P->rplimit = 0;
     proc_P = Remove_Process (node_P, proc_P->upid);
     if (proc_P != NULL)
     {
      migrate (node_P, proc_P, msg_P->sender_id);
      node_P->n_local_procs--;
     }
    }
   }
  }
}
```

```
/*************************************************************/
/*                                                           */
/*   G B A L _ R E P L Y _ M S G                             */
/*                                                           */
/*************************************************************/
void   Gbal_reply_msg (PROCESSOR *node_P, MSG_FRAME *msg_P)
{
 USER_PROCESS        *proc_P;

 /* received by CPU(i) */
 switch (msg_P->service_no)
 {
 case PRB_RPLY_MSG:
      if (node_P->high_t_set &&
         (msg_P->ack_reply < FALSE))
      {
        Cancel_Timer(HIGH_TOUT, node_P, NULL);
        Cancel_Timer(RCV_AVG_TOUT, node_P, NULL);
        node_P->high_t_set = 0;
        proc_P = longest_runner (node_P);
        if (proc_P != NULL)
        {
         migrate (node_P,
                Remove_Process (node_P, proc_P->upid),
                msg_P->sender_id);
         node_P->n_local_procs--;
        }
      }
      break;


 case WK_RPLY_MSG:
      if (node_P->low_t_set
         && (msg_P->ack_reply > FALSE))
      {
         Cancel_Timer(LOW_TOUT, node_P, NULL);
         /* Cancel_Timer(RCV_AVG_TOUT, node_P, NULL); */
         node_P->timq.itype = WAITP_TOUT;
         threshold = node_P->threshold;
         work_reply_msg (node_P, msg_P);
         node_P->low_t_set = 0;
      }
 }

}
```

```
/******************************************************************/
/*                                                                */
/*    T R A N S F E R _ M S G                                     */
/*                                                                */
/******************************************************************/
void   transfer_msg (PROCESSOR *node_P, struct comm_packet *msg_P)
{
 USER_PROCESS  *proc_P;

/* received by CPU(i)
 if (node_P->rplimit)
 {
   node_P->rplimit--;
   proc_P = msg_P->mess_data;
   Cancel_Timer(SRQ_TOUT, node_P, proc_P);
   if (msg_P->ack_reply < TRUE)
   {
     proc_P = Remove_Process (node_P, proc_P->upid);
     if (proc_P != NULL)
     {
       migrate (node_P, proc_P, msg_P->sender_id);
       node_P->n_local_procs--;
     }
   }
   else node_P->n_active_local_procs++;
   Select_next_timeout(node_P);
 }
 else
*/
    if ((msg_P->ack_reply < TRUE)
       && (node_P->n_local_procs
          > node_P->threshold + N_SYS_PROCS))
    {
        proc_P = longest_runner(node_P);
        if (proc_P != NULL)
        {
         migrate (node_P,
                Remove_Process (node_P, proc_P->upid),
                msg_P->sender_id);
         node_P->n_local_procs--;
        }
     }
}
```

```
/******************************************************************/
/*                                                                */
/*    P R O C E S S _ M S G                                       */
/*                                                                */
/******************************************************************/
void   process_msg (PROCESSOR *node_P,MSG_FRAME *msg)
{
 USER_PROCESS        *proc_P;

 proc_P = msg->mess_data;
 proc_P->exec_here_time = 0;
 proc_P->residency_time = 0;
 proc_P->exist_time   += msg->distance
                    * (PROTOCOL_TIME * AVE_INST
                    + msg->total_bytes * TX_BYTE_TIME);
 proc_P->schedulable   = TRUE;
 proc_P->nxt_proc_ptr  = NULL;
 (void) Add_Process (proc_P->upid, proc_P, node_P);
 node_P->n_immigs++;
 time_update (node_P, (int) TIME_RCV_PROC, OStime);

#ifdef DEBUG_MSG
    if ((process_count >= MIN_PID)
       && (process_count <= MAX_PID))
      fprintf (monitor_file,
           "\nnode %d RECEIVES process%d [load = %d] FROM node %d\n",
           node_P->node_id,
           proc_P->upid,
           node_P->load, msg->sender_id);
#endif

 if (node_P->n_virtual_procs > 0)
 {
  node_P->n_virtual_procs--;
  if (node_P->n_virtual_procs <= 0)
    Cancel_Timer(WAITP_TOUT, node_P, NULL);
 };

}


/*************************/
/*                       */
/*    E X I T _ M S G    */
/*                       */
/*************************/
void   exit_msg (PROCESSOR *nodeP, MSG_FRAME *msg)
{
 /* Do nothing */
 nodeP->n_deaths++;
 nodeP->cumul_exist_time += msg->ldvalue;

}
```

```
/*****************************************************************/
/*                                                               */
/*    E X I T _ P R O C                                          */
/*                                                               */
/*****************************************************************/
void   exit_proc ( PROCESSOR *node_P, MSG_FRAME *msg)
{
 ROUTE_ENTRY          next_route;
 USER_PROCESS         *proc_P;
 extern void          exit_dump(PROCESSOR *, USER_PROCESS *);

 context_switch;
 time_update(node_P,
        (int) (CONTEXT_SWITCH * AVE_INST + TIME_EXIT * AVE_INST),
        USERtime);
 proc_P = msg->mess_data;
 if (proc_P->orig_mc != node_P->node_id)
 {
   /* set up exit msg for original mc */
   /* send msg */
   /* TX (sizeof(EM_EXIT, OStime); */
   next_route  = route_node (proc_P->orig_mc,
                   node_P->links,
                   node_P->route_table);
   dblock      =  pack_data (proc_P, PSTOP_MSG, 4);
   dblock->ldv =  proc_P->exist_time;
   transmit (node_P,
          next_route,
          create_frame (node_P, proc_P->orig_mc));
   free ((char *)dblock);
 }
 else
 {
  node_P->cumul_exist_time += proc_P->exist_time;
  node_P->n_deaths++;
 }
 exit_dump(node_P, proc_P);
 /* MARK process for removal */
 proc_P->finished = TRUE;
 proc_P->schedulable = FALSE;
 node_P->n_local_procs--;
 node_P->n_active_local_procs--;
}
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  This is the the computational module, encompassing the accumulation
  of basic statistics such as averages and totals.

  It differs from the original by encompassing "ave_compute_or_display(),
  and the "job_creation() modules also.

  The job_creation module now provides a different randomizing seed
  each time the job is run. AND,

  Convergence is represented purely in terms of the percentage diff.
  in runtime between the fastest and slowest processor.

        Modified:       5th December 1991
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "model_params.h"
#include "model_types.h"
#include "globvar.h"

int             Least_loaded,
                last_t,
                Most_loaded,
                sigma_diff;

float           Mean_load,
                sigma_load,
                sigma_var,
                sigma_rtime,
                sigma_migrate,
                sigma_Tx,
                Mprofile,
                Ave_migration(void),
                Ave_transmission(void);
extern float    Estimator();
extern  int     njobs,
                probe_limit,
                process_count,
                total_pe;
extern float    load_value,
                threshold;
extern  int         lba_index;
extern double          global_clock;

extern FILE         *trace,
                       *seedfile,
               *monitor_file,
               *configfile,
               *paramfile;

extern NODE_ENTRY    *processor_table;
extern int          Sec, Del;
```

```
/***************************************************************/
/*                                                             */
/*    a v e _ c o m p u t e _ o r _ d i s p l a y              */
/*                                                             */
/***************************************************************/
void ave_compute_or_display (BOOLEAN run_complete)
{

 if (run_complete)
 {
  float      tsec;

  tsec = global_clock / ONE_SECOND;
  fprintf(monitor_file,  "\n%8.2f\t", tsec);
  fprintf(monitor_file, "%6.4f\t", sigma_var / tsec);
  fprintf(monitor_file, "%6.4f\t", sigma_load / tsec);
  fprintf(monitor_file, "%6.4f\t", sigma_diff / tsec);
  fprintf(monitor_file, "%6.4lf\t", Mprofile/tsec);
  fprintf(monitor_file, "%7.2f\t",
         100 * data_convergence((int) tsec));
  fprintf(monitor_file, "%7.4f\t", Ave_migration() / tsec);
  fprintf(monitor_file, "%7.4f\t",  Ave_transmission()/ tsec);
  fprintf(monitor_file, "%5.2f\t\n", 100.0 * process_count/ njobs);
 }
 else
  {
 /*   fprintf (trace, "\t%2.2f\t%d\t%2.2f\t%2.2f\t%2.2f\t%2.2f\n",
  sigma_load / tsec,
  sigma_diff / tsec,
  sigma_var / tsec,
  Mprofile/tsec,
  sigma_migrate / tsec,
  sigma_Tx / tsec);
  */
  }
#ifdef DEBUG_MSG
  Dump_RTime();
#endif
}
```

```
/****************************************************************/
/*                                                              */
/*   f i n a l _ c o m p u t e                                  */
/*                                                              */
/****************************************************************/

void final_compute (void)
{


  float      tsec;

   tsec = global_clock / ONE_SECOND;
  fprintf(monitor_file,"\n%8.2f\t%6.4f\t%6.4f\t%6.4f\t%6.4f\t%7.2f\t%7.4f\t%7.4f\t%5.2f\t\n",
    tsec,
    sigma_var / tsec,
    sigma_load / tsec,
    sigma_diff / tsec,
    Mprofile/tsec,
    100 * data_convergence((int) tsec),
    Ave_migration() / tsec,
    Ave_transmission() / tsec,
    100.0 * process_count/ njobs);

#ifdef DEBUG_MSG
    Dump_RTime();
#endif
  }
```

```
/***************************************************************/
/*                                                             */
/*   M E A N _ D I F F E R E N C E                             */
/*                                                             */
/***************************************************************/
float   mean_difference (void)
{
 PROCESSOR          *nodeP;
 int                total_load,
                    idx;

 Least_loaded  = 1000;
 Most_loaded   = 0;
 total_load    = 0;
 for (idx = 0; idx < total_pe; idx++)
 {
/*
    fprintf (trace, "\n");
*/
    nodeP  = processor_table[idx].node_adr;
/*
     fprintf (trace, "Node%d  Load = %d  ",
            nodeP->node_id, nodeP->n_local_procs);
*/

     if (nodeP->load < Least_loaded)
       Least_loaded = nodeP->load;
     if (nodeP->load > Most_loaded)
       Most_loaded  = nodeP->load;
     total_load       += nodeP->load;
/*
    fprintf (trace, "\n");
*/
 }
 return ((float)total_load / total_pe);
}

/***************************************************************/
/*                                                             */
/*   V A R I A N C E                                           */
/*                                                             */
/***************************************************************/
float   variance (float mean)
{
 PROCESSOR          *nodeP;
 float              sum_var,
                    diff;
 int                idx;

 sum_var      = 0.0;

 for (idx = 0; idx < total_pe; idx++)
 {
   nodeP  = processor_table[idx].node_adr;
   diff          = ((float)nodeP->load) - mean;
   sum_var       += (diff * diff);
 }
 return (sum_var / (total_pe - 1));
}
```

```
/*****************************************************************/
/*                                                               */
/*    P R O C E S S _ R U N T I M E                              */
/*                                                               */
/*****************************************************************/
float   Process_runtime (void)
{
 PROCESSOR          *nodeP;
 float              sum_rtime,
                    rtime;

 int                idx;

 sum_rtime      = 0.0;

 for (idx = 0; idx < total_pe; idx++)
 {
    nodeP  = processor_table[idx].node_adr;
     if (nodeP->n_deaths != 0)
     {
       rtime = nodeP->cumul_exist_time / nodeP->n_deaths;
       sum_rtime += (rtime / ONE_SECOND);
     }
 }
 return (sum_rtime / total_pe);
}


/*****************************************************************/
/*                                                               */
/*    A V E _ M I G R A T I O N                                  */
/*                                                               */
/*****************************************************************/

float   Ave_migration (void)
{
 PROCESSOR          *nodeP;
 float              sum_migrate;

 int                idx;

 sum_migrate      = 0.0;

 for (idx = 0; idx < total_pe; idx++)
 {
    nodeP    = processor_table[idx].node_adr;
    sum_migrate += nodeP->n_migrates;
 }
 return (sum_migrate / total_pe);
}
```

```
/****************************************************************/
/*                                                              */
/*   A V E _ T R A N S M I S S I O N                            */
/*                                                              */
/****************************************************************/

float   Ave_transmission (void)
{
 PROCESSOR        *nodeP;
 float            sum_tx;

 int              idx;

 sum_tx     = 0.0;

 for (idx = 0; idx < total_pe; idx++)
 {
    nodeP  = processor_table[idx].node_adr;
    sum_tx   += nodeP->n_tx;
 }
 return (sum_tx / total_pe);
}



/****************************************************************/
/*                                                              */
/*   S I M U L A T I O N _ C O M P L E T E                      */
/*                                                              */
/****************************************************************/
BOOLEAN   simulation_complete(void)
{
 BOOLEAN      any_queue_empty();
 float        data_convergence(int);
 int            i, tsec;

 tsec = (int)(global_clock / ONE_SECOND);
 i   = tsec % SAMPLE_T;
 if (i != last_t)
 {
  if (any_queue_empty())
    return QUEUES_FLUSHED;
  Mprofile    += Process_runtime();
  Mean_load   = mean_difference ();
  sigma_load  += Mean_load;
  sigma_diff  += (Most_loaded - Least_loaded);
  sigma_var   += variance(Mean_load);
 }
 if ((i != last_t) && (tsec > STABLE_STATE))
   if ((tsec % 10) == 0)
   {
    if (data_convergence(tsec) < CONVERGE_FACTOR)
      return DATA_CONVERGED;
    else if ((Sec > 0) && (process_count/Sec > Del/Sec))
         return QUEUES_FLUSHED;
   }
 last_t = i;
 return FALSE;
}
```

```
/****************************************************************/
/*                                                              */
/*   d a t a _ c o n v e r g e n c e                            */
/*                                                              */
/****************************************************************/
float    data_convergence (int tsec)
{
 float      rtime,
                 G_avg;

 rtime   = Process_runtime();
 G_avg   = Mprofile/tsec;

 if (rtime > G_avg)
    return  (rtime - G_avg) / rtime;
 else
    return  (G_avg - rtime) / G_avg;
}


/****************************************************************/
/*                                                              */
/*   T I M E _ U P D A T E                                      */
/*                                                              */
/****************************************************************/

void time_update (PROCESSOR *nodeP, int elapsed_time, int time_type)

{
 void          quanta_update(PROCESSOR *, int, int);
 float         calc_ave_load(PROCESSOR *);

 USER_PROCESS   *process_ptr;

 nodeP->sys_real_time += elapsed_time;
 if (time_type == USERtime)
   {
    nodeP->curr_process->exec_time       += elapsed_time;
    nodeP->curr_process->exec_here_time  += elapsed_time;
   }
/*
 printf ("\n Time update P%d exec_time %lf\n",
        nodeP->curr_process->upid,
        nodeP->curr_process->exec_time);
*/

 process_ptr   = nodeP->firstjb_ptr;
 while (process_ptr != NULL)
 {
  process_ptr->exist_time += elapsed_time;
  process_ptr            = process_ptr->nxt_proc_ptr;
 }
 quanta_update (nodeP, elapsed_time, time_type);
 nodeP->load = (int) calc_ave_load (nodeP);

}
```

```
/****************************************************************/
/*                                                              */
/*    Q U A N T A _ U P D A T E                                 */
/*                                                              */
/****************************************************************/
void quanta_update (PROCESSOR *nodeP, int elapsed_time, int time_type)
{
 int added_time,
     idx;

 void next_quantum(PROCESSOR *),
     add_to_quantum(PROCESSOR *, int, int);


 idx = nodeP->quanta_idx;

 while (elapsed_time > 0)
 {
    added_time = elapsed_time;
    if (elapsed_time + nodeP->quanta[idx].used >= QUANTUM)
    {
     added_time = QUANTUM - nodeP->quanta[idx].used;
     add_to_quantum (nodeP, added_time, time_type);
     next_quantum(nodeP);
    }
    else
     add_to_quantum (nodeP, added_time, time_type);

    elapsed_time -= added_time;
 }
}


/****************************************************************/
/*                                                              */
/*    A D D _ T O _ Q U A N T U M                               */
/*                                                              */
/****************************************************************/
void add_to_quantum (PROCESSOR *nodeP, int add_time, int time_type)
{
 int  idx;

 idx  = nodeP->quanta_idx;

 if (time_type == OStime)
 {
    nodeP->quanta[idx].OSportion += add_time;
    nodeP->OSoverhead           += add_time;
 }

 nodeP->quanta[idx].used     += add_time;
}
```

```
/**************************************************************/
/*                                                            */
/*    N E X T _ Q U A N T U M                                 */
/*                                                            */
/**************************************************************/
void next_quantum (PROCESSOR *nodeP)
{
 int  idx;

 nodeP->quanta_idx++;
 if (nodeP->quanta_idx >= NQUANTA)
 {
   nodeP->quanta_idx = 0;
 }

 idx                          = nodeP->quanta_idx;
 nodeP->quanta[idx].used        = 0;
 nodeP->OSoverhead             -= nodeP->quanta[idx].OSportion;
 nodeP->quanta[idx].OSportion   = 0;
 nodeP->quanta[idx].actual_load  = nodeP->n_local_procs;
 nodeP->quanta[idx].virtual_load = nodeP->n_virtual_procs;
}

/**************************************************************/
/* For  Dynamic  Load   Balancing                             */
/*    C A L C _ A V E _ L O A D                               */
/*                                                            */
/**************************************************************/
float dyn_calc_ave_load(PROCESSOR *nodeP)
{
  int total_load,
      idx;

      total_load = 0;

      for (idx=0; idx <NQUANTA; idx++)
         total_load += nodeP->quanta[idx].actual_load
                    + nodeP->quanta[idx].virtual_load;
      return ((float) total_load/NQUANTA);
}

/**************************************************************/
/*                                                            */
/*    C A L C _ A V E _ L O A D                               */
/*                                                            */
/**************************************************************/
float calc_ave_load(PROCESSOR *nodeP)
{
  int total_load,
      idx;

      total_load = 0;

      for (idx=0; idx <NQUANTA; idx++)
         total_load += nodeP->quanta[idx].actual_load;
      return ((float) total_load/NQUANTA);
}
```

```
unsigned short exp_xsubi[] = {55213, 10232, 2721};
unsigned short rnd_xsubi[] = {3, 4, 5};
int            process_groups[] = {2, 3, 4};
unsigned short  rnd_job[]       = {7, 8, 9};

void job_creation (short int buffer_no)
{
 extern float  load_value;
 extern int    njobs,
               total_pe;

  char       **io_buffer_blk;
  int         i;
  char        job_name[16];
  double      rnd_num,
               rnd_grp,
               new_rand;
  FILE       **jobfile_blk;

  static double      t      = 0.0;
  static int         start  = 0,
                     finish = 0;
  int                pgsize = PAGE_SIZE;

 extern void   setbuf(FILE *, char  *);
 extern double log(double);
 extern double erand48(unsigned short []),
                   drand48(void);

 if (buffer_no == 0)
 {
         finish = 0;
         t      = 0.0;
         start  = 0;
     if (njobs < PAGE_SIZE)
       pgsize = njobs;
         if (seedfile == NULL)
         {
          /**************************************************************
                   Code to randomize jobstreams for each simulation run
          ************************************************************/
          srand48 ((unsigned) time (NULL));
          for (i = 0; i < 3 ; i++)
          {
                  exp_xsubi[i] = (int)(drand48() * 65535);
          }
          /*------------------------------------------------------------*/
        }
        else
          {
           fscanf(seedfile, "%hu\n", &exp_xsubi[0]);
           fscanf (seedfile, "%hu\n", &exp_xsubi[1]);
           fscanf(seedfile, "%hu\n", &exp_xsubi[2]);
          };
        fprintf (monitor_file,
                 "\n *** erand48 seeds are: %d, %d, and %d  ***\n",
                 exp_xsubi[0], exp_xsubi[1], exp_xsubi[2]);
};
```

```
finish += pgsize;
if (finish > njobs)
        return;

 io_buffer_blk = (char **) calloc (total_pe, sizeof(char *));

jobfile_blk = (FILE **) calloc (total_pe, sizeof(FILE *));
/* open job files */
for (i= 0; i < total_pe; i++)
{
  *(io_buffer_blk + i) = calloc (BUFSIZ, sizeof(char));

  sprintf (job_name, "jobs%d_pg%d", i, buffer_no);
  *(jobfile_blk + i) = fopen(job_name, "wb");

   setbuf(*(jobfile_blk + i), *(io_buffer_blk + i));

};

/* create job streams */
for (; start < finish; start++)
{
     /* Generate exp. random number. */
     rnd_num  = erand48(exp_xsubi);
     new_rand = -log(rnd_num)/((double)(load_value * total_pe));

     /* establish time of creation */
     t = t + new_rand * 1000000 * AVE_PROC_GROUP * AVE_EXEC_TIME;

     /* give job to random mc */
     rnd_grp = erand48(rnd_job);
     rnd_grp = rnd_grp * 3;
     rnd_num = erand48(rnd_xsubi);
     rnd_num = rnd_num * total_pe;

     fwrite ((const char *) &process_groups[(int)rnd_grp],
            sizeof(process_groups[(int)rnd_grp]),
            1,
            *(jobfile_blk + (int)rnd_num));
     fwrite ((const char *)&t,
            sizeof(t),
            1,
            *(jobfile_blk + (int)rnd_num));


  }

  /* close job files */
  for (i= 0; i < total_pe; i++)
  {
    cfree (*(io_buffer_blk + i));
   fclose (*(jobfile_blk + i));
  }
  cfree ((char *) jobfile_blk);
  cfree ((char *) io_buffer_blk);
}
```

```
/* ============================================================= */
/*     This version of initb_module on SparcStation One has       */
/*     been modified with:                                        */
/*        a) Routine int_divisible - double/int division &        */
/*           casting of objects.                                  */
/*        b) Efficient version of all_queues_empty - if any       */
/*           job stream dries up, simulation terminates!          */
/*        c) Data_has_converged routine made more                 */
/*           computationally efficient by computing average       */
/*        time once only!                                         */
/*     d) Further efficiency improvement by finding the           */
/*           minimum and maximum run times only.                  */
/*                                                                */
/*     Date of Modification: 5th December 1991                   */
/* ============================================================= */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <math.h>
#include <time.h>
#include <strings.h>
#include "model_params.h"
#include "model_types.h"
#include "globvar.h"

/****************************************************************/
/*    CONSTANT, GLOBAL, & LITERAL                                */
/*         DEFINITIONS.                                          */
/****************************************************************/


extern int          SkipRun;
extern FILE        *monitor_file;

extern NODE_ENTRY     *processor_table;
extern int         njobs,
                   process_count,
                   total_pe;

extern double        global_clock;
extern char        **io_buffer_blk;

extern void job_creation (short int);
extern double erand48(unsigned short []);


  void          MsgQ_Dump(MESS_QUEUE),
              Dump_Msg(MSG_FRAME *);
```

```
/*******************************************/
/*                                         */
/*    W R I T E _ T O _ M S G Q            */
/*                                         */
/*******************************************/
void Write_To_MsgQ(MESS_QUEUE *queue, MSG_FRAME *msgP)
{
  MSG_FRAME      *curr_message,
                 *prev_message;

  if (queue->head == NULL)
  {
   queue->head = msgP;
   queue->tail = msgP;
  }
  else
   {
     prev_message = queue->head;
     curr_message = prev_message;
     while (curr_message != NULL)
      if (msgP->time_stamp >= curr_message->time_stamp)
      {
       prev_message = curr_message;
       curr_message = curr_message->next_pack_ptr;
      }
      else break;

     if (curr_message == queue->head)
     {
       msgP->next_pack_ptr = curr_message;
       queue->head      = msgP;
     }
     else
      {
       if (curr_message == NULL)
       {
          queue->tail = msgP;
       };
       prev_message->next_pack_ptr = msgP;
       msgP->next_pack_ptr= curr_message;
      }
  }
 /* NodeQ_Dump(nodeP); */

}
```

```
/****************************************************************/
/*                                                              */
/*    R E A D _ F R O M _ M S G Q                               */
/*                                                              */
/****************************************************************/
MSG_FRAME *Read_From_MsgQ (MESS_QUEUE *queue)
{
   MSG_FRAME  *msg_P;

  msg_P   = NULL;
  if (queue->head != NULL)
  {
     msg_P              = queue->head;
     queue->head        = queue->head->next_pack_ptr;
     msg_P->next_pack_ptr = NULL;
     if (queue->head == NULL)
       queue->tail = NULL;
  };
  return (msg_P);
}


void Delete_Cset (PROCESSOR *nP, int id)
{
 int        i = 0;

 while (i < CSET_SIZE)
 {
    if (nP->cSet[i].id < 0)
              break;
    if (nP->cSet[i].id == id)
    {
       while (i < CSET_SIZE - 1)
       {
         nP->cSet[i].id = nP->cSet[i + 1].id;
         nP->cSet[i].ld = nP->cSet[i + 1].ld;
         i++;
       };
       nP->cSet[i].id = -1;
       nP->cSet[i].ld = 0;
       return;
    }
    else i++;
 };
 return;
}
```

```
void Update_Cset ( PROCESSOR *nP, MSG_FRAME *msg, int sort)
{
  int       ld,
            wtLd,
            sgn;
  BOOLEAN     sorted;

  sorted = FALSE;
  Delete_Cset (nP, msg->sender_id);
  ld   = msg->ack_reply;
  if  (ld < 0)
    sgn = -1;
  else
    sgn = 1;
  wtLd = ld ;  /* weighted : (int) ((ld*2)/msg->distance + 0.5); */
  wtLd *= sgn;
  {
    int       dv[2],
              p, q, i, j;
    if  (sort == DESCEND)
    {
      p = 0; q = 1;
    }else
    {
      p = 1; q = 0;
    }
    i = 0;
    while ((i < CSET_SIZE)  && (!sorted))
    {
      if  (nP->cSet[i].id < 0)
      {
        sorted = TRUE;
      }
      else
      {
       dv[1] = nP->cSet[i].ld;
       dv[0] = wtLd;
       if (dv[q] < dv[p])
       {
        j = CSET_SIZE - 1;
        while (j  >  i)
        {
         nP->cSet[j].id = nP->cSet[j - 1].id;
         nP->cSet[j].ld = nP->cSet[j - 1].ld;
         j--;
        }
        sorted = TRUE;
       } else i++;
      }
    };
   if (sorted)
   {
    nP->cSet[i].id = msg->sender_id;
    nP->cSet[i].ld = wtLd;
   }
  }
#ifdef DEBUG_MSG
 if ((process_count >= MIN_PID)
    && (process_count <= MAX_PID))
```

```
 {
   fprintf (monitor_file, "\n\tComms set:\tNode\tLoad\n");
   ld = 0;
   while (ld < CSET_SIZE)
   {
    fprintf (monitor_file, "\tN%d\t\t%d\t%d\n",
          nP->node_id, nP->cSet[ld].id, nP->cSet[ld].ld);
    ld++;
    }
  }
#endif
   return;
}


/*************************************************************/
/*                                                           */
/*    D U M P _ C S E T                                      */
/*                                                           */
/*************************************************************/
void Dump_Cset ( PROCESSOR   *pe)
{

   int ld = 0;

   fprintf (monitor_file, "[");
   while ((ld < CSET_SIZE) && (pe->cSet[ld].id >=0))
   {
    fprintf (monitor_file, "%d->%d ",
          pe->cSet[ld].id, pe->cSet[ld].ld);
    ld++;
    }
   fprintf (monitor_file, "]\n\n");
}

/*************************************************************/
/*                                                           */
/*    C S E T _ L O A D _ A V G                              */
/*                                                           */
/*************************************************************/
int Cset_Load_Avg (PROCESSOR   *pe)
{  int ld, total = 0, count = 0;

   ld    = (int) pe->threshold;
   while ((count < CSET_SIZE) && (pe->cSet[count].id >=0))
   {
    count++;
    total = total + pe->cSet[count].ld;
   }
   if (count > 0)
   {
     total = total/count;
     if (total > ld)
        return ld;
    }
    if (ld > 2)
      return ld - 1;
    else return ld;
}
```

```
/****************************************************************/
/*                                                              */
/*    D U M P _ M S G                                           */
/*                                                              */
/****************************************************************/
void Dump_Msg (MSG_FRAME  *msg_P)
{

   fprintf ( monitor_file,
         "\tSender%d    Dest%d    distance%d   Arr_Time:%f",
         msg_P->sender_id, msg_P->dest_id, msg_P->distance,
         msg_P->time_stamp / ONE_SECOND);
   fprintf (monitor_file, "\n\tType%d   ServNo%d   Reply%d\n\n",
        msg_P->type, msg_P->service_no, msg_P->ack_reply);

}

/****************************************************************/
/*                                                              */
/*    M S G Q _ D U M P                                         */
/*                                                              */
/****************************************************************/
void MsgQ_Dump ( MESS_QUEUE queue)
{
   MSG_FRAME  *msg_P;

  msg_P   = queue.head;
  while (msg_P != NULL)
  {
   Dump_Msg (msg_P);
   msg_P   = msg_P->next_pack_ptr;
  };
}


/****************************************************************/
/*                                                              */
/*    N O D E Q _ D U M P                                       */
/*                                                              */
/****************************************************************/
void NodeQ_Dump (PROCESSOR  *nodeP)
{
  fprintf (monitor_file, "\nMsgQ State for node%d at time:%f\n",
        nodeP->node_id, nodeP->sys_real_time / ONE_SECOND);
  MsgQ_Dump (nodeP->queue);
}
```

```
/*****************************************************************/
/*                                                               */
/*    D u m p _ R T i m e                                        */
/*                                                               */
/*****************************************************************/
void        Dump_RTime (void)
{
  int       Check_for_Zero (int , int),
            i;
  float           rtime;
  PROCESSOR    *nodeP;

  for (i = 0; i < total_pe ; i++)
  {
    nodeP      = processor_table[i].node_adr;
    rtime      = nodeP->cumul_exist_time
                 / nodeP->n_deaths;
    fprintf (monitor_file,
          "N:%4d\tMg:%6.2f\tL:%4d\tRt:%6.2f\tTx:%6d  Imm:%4d  Dth:%6d",
        i,
        (float)nodeP->n_migrates,
        nodeP->n_local_procs,
       rtime/ONE_SECOND,
        Check_for_Zero(nodeP->n_tx,nodeP->n_immigs),
        nodeP->n_immigs,
        nodeP->n_deaths);
/*
    fprintf (monitor_file, "\nCommSet:");
    Dump_Cset(nodeP);
*/
  }
  fprintf(monitor_file, "\n\n");
}

int Check_for_Zero (int x, int y)
{
  if (y == 0)
    return x;
  else
    return x/y;
}
```

```
/*********************************************/
/*                                           */
/*   E R A S E _ F R O M _ M S G Q          */
/*                                           */
/*********************************************/
MSG_FRAME *Erase_From_MsgQ(MESS_QUEUE *queue,
                int msg_type,
                USER_PROCESS *prc)
{
  MSG_FRAME      *last_msg,
             *this_msg;

  last_msg = this_msg = queue->head;
  do
  {
   if (this_msg == NULL)
    return (NULL);
   else
    if (this_msg->service_no == msg_type)
     {
      if ((prc == NULL) || (this_msg->mess_data == prc))
      {
      if (last_msg == this_msg)
      {
        queue->head = this_msg->next_pack_ptr;
        if (this_msg == queue->tail)
          queue->tail = queue->head;
       }
      else
        last_msg->next_pack_ptr = this_msg->next_pack_ptr;
      this_msg->next_pack_ptr = NULL;
      if (this_msg == queue->tail)
        queue->tail = last_msg;
        free ((char *) this_msg);
      return (this_msg);
       }
    }
    last_msg = this_msg;
    this_msg = this_msg->next_pack_ptr;

  } while (TRUE);
}
```

```
/*****************************************************************/
/*                                                               */
/*   R E S T A R T _ C L O C K                                   */
/*                                                               */
/*****************************************************************/
void restart_clock(PROCESSOR *nodeP,
            int itype,
            double max_time,
            USER_PROCESS *prc)
{
 double      time_stamp;
 char        *Tim_Text(int);
 BOOLEAN     reset;


 time_stamp  = nodeP->sys_real_time + max_time;
 reset = (!nodeP->timq.itype)
       || ((nodeP->timq.itype != WAITP_TOUT) && (prc == NULL))
       || ((nodeP->timq.itype) && (time_stamp < nodeP->timq.mintime))
       || ((nodeP->timq.itype == WAITP_TOUT) && (itype == WAITP_TOUT)
          && (time_stamp > nodeP->timq.mintime));
 if (reset)
 {
   nodeP->timq.mintime = time_stamp;
   nodeP->timq.itype  = itype;
   if (itype == SRQ_TOUT)
     nodeP->timq.minP   = prc;
 }
 if (prc != NULL)
 {
   prc->timeout = time_stamp;
   prc->itype  = itype;
 }
#ifdef DEBUG_MSG
 if ((process_count >= MIN_PID)
    && (process_count <= MAX_PID))
   fprintf (monitor_file,
        "\nNode%d GMT%8.3f \tTripT%4.3f \tTIMER:%s\t Etime:%8.3f\n",
        nodeP->node_id,
        nodeP->sys_real_time / ONE_SECOND,
        max_time / ONE_SECOND,
        Tim_Text(itype),
        time_stamp / ONE_SECOND);
#endif

}
```

```
/*****************************************************************/
/*                                                               */
/*  S E L E C T _ N E X T _ T I M E O U T                        */
/*                                                               */
/*****************************************************************/
void Select_next_timeout(PROCESSOR *nodeP)
{
  USER_PROCESS  *proc;

  proc      = nodeP->firstjb_ptr;

  nodeP->timq.itype = 0;
  nodeP->timq.minP   = NULL;
  while (proc != NULL)
  {
   if (proc->timeout)
   {
     nodeP->timq.mintime = proc->timeout;
     nodeP->timq.itype   = proc->itype;
     nodeP->timq.minP    = proc;
     break;
   }
   else
     proc = proc->nxt_proc_ptr;
  }

  while (proc != NULL)
  {
   if ((proc->timeout) &&
      (proc->timeout < nodeP->timq.mintime))
   {
     nodeP->timq.mintime = proc->timeout;
     nodeP->timq.itype   = proc->itype;
     nodeP->timq.minP       = proc;
   };
   proc = proc->nxt_proc_ptr;
  }
}
```

```
/* ============================================================ */
/*    T i m _ T e x t                                           */
/*                                                              */
/* ============================================================ */
char   *Tim_Text(int tcode)
{
  switch (tcode)
  {
   case RCV_AVG_TOUT:
             return ("RCV_AVG_TOUT\0");
   case HIGH_TOUT: return ("HIGH_TOUT\0");
   case LOW_TOUT : return ("LOW_TOUT\0");
   case SRQ_TOUT : return ("SRQ_TOUT\0");
   case PRB_TOUT : return ("PRB_TOUT\0");
   case WAITP_TOUT: return ("WAITP_TOUT\0");
   default: return ("Timer UNDEFINED\0");
  }/* switch */
}

/******************************************/
/*                                        */
/*   P A G E _ F A U L T I N G            */
/*                                        */
/******************************************/
BOOLEAN   page_faulting (PROCESSOR *nodeP)
{
 short int         node_id;
 FILE              *tempf_ptr;
 char              prev_fname[16];

 /* if (!positive_response) */
    return FAILED;
  node_id    = nodeP->node_id;
  sprintf (prev_fname, nodeP->event_fname);
  tempf_ptr  = nodeP->job_file;
  sprintf (nodeP->event_fname, "jobs%d_pg%d",
        node_id, nodeP->buffer_no + 1);
  if (init_eventfile (nodeP, node_id) == FAILED)
  {
    job_creation (nodeP->buffer_no + 1);
    if (init_eventfile (nodeP, node_id) == SUCCESS)
    {
     (nodeP->buffer_no)++;
      fclose (tempf_ptr);
      unlink (prev_fname);
    }else
     {
      nodeP->job_file    = tempf_ptr;
     sprintf (nodeP->event_fname, prev_fname);
     }
  } else
    {
    (nodeP->buffer_no)++;
     fclose (tempf_ptr);
    unlink (prev_fname);
    }
 return SUCCESS;
}
```

```
/********************************************/
/*                                          */
/*   INIT_EVENTFILE                         */
/*                                          */
/********************************************/
int    init_eventfile(PROCESSOR *nodeP, short int node_id)
{
  int              nxt_job;
  double     nxt_arrival_time;

   if ((nodeP->job_file = fopen(nodeP->event_fname, "rb"))
    == NULL) return FAILED;
   else
    {
     setbuf(nodeP->job_file, *(io_buffer_blk + node_id));
     fread ((char *)&nxt_job, sizeof(nxt_job),
          1,
          nodeP->job_file);
     fread ((char *)&nxt_arrival_time, sizeof(nxt_arrival_time),
          1,
          nodeP->job_file);
     process_count++;
     nodeP->nxt_arr_time = nxt_arrival_time;
     return SUCCESS;
    }
}
```

```
/*******************************************/
/*                                          */
/*    C L O S E _ E V E N T F I L E S       */
/*                                          */
/*******************************************/
void   close_eventfiles (void)
{
 int              idx;

 PROCESSOR        *shadow_node;

 for (idx = 0; idx < total_pe; idx++)
 {
  shadow_node = processor_table[idx].node_adr;
  fclose(shadow_node->job_file);
  { /* flush process queue */
   USER_PROCESS  *process_ptr,
                 *del_prc;

   process_ptr  = shadow_node->firstjb_ptr;
   while (process_ptr != NULL)
   {
    del_prc     = process_ptr;
    process_ptr = process_ptr->nxt_proc_ptr;
    free ((char *) del_prc);
   };
  }
  { /*  flush message queue */
   MSG_FRAME     *msg_P,
                 *del_msg;

   msg_P   = shadow_node->queue.head;
   while (msg_P != NULL)
   {
    del_msg    = msg_P;
    msg_P      = msg_P->next_pack_ptr;
    free ((char *) del_msg);
   };
  }

#ifdef DEBUG_MODEL
   fclose(shadow_node->trace_file);
#endif
   cfree (*(io_buffer_blk + idx));
 }
 cfree ((char *) (io_buffer_blk));
}
```

```
/****************************************************************/
/*                                                              */
/*   P R O C  Q _ L I S T                                       */
/*                                                              */
/****************************************************************/
void   procQ_List (PROCESSOR  *nodeP)
{
  USER_PROCESS        *proc_P;
  int                 length;

  length = 0;
  proc_P  = nodeP->firstjb_ptr;
  fprintf (monitor_file,"\nNode%d PQ: ", nodeP->node_id);
  while (proc_P != NULL)
  {
    if (length++ % 10);
    else fprintf(monitor_file,"\n");
    fprintf (monitor_file, "pid:%d, ", proc_P->upid);
    proc_P = proc_P->nxt_proc_ptr;
  }
  fprintf(monitor_file,"\n");
}

/****************************************************************/
/*                                                              */
/*   P R O C  Q _ L E N G T H                                   */
/*                                                              */
/****************************************************************/
int   ProcQ_Length (USER_PROCESS *queue)
{
  int      length;

  length = 0;
  while (queue != NULL)
  {
    length++;
    queue = queue->nxt_proc_ptr;
  }
  return (length);
}


/****************************************************************/
/*                                                              */
/*   M S G  Q _ L E N G T H                                     */
/*                                                              */
/****************************************************************/
int   MsgQ_Length (MSG_FRAME  *queue)
{
  int      length;

  length = 0;
  while (queue != NULL)
  {
    length++;
    queue = queue->next_pack_ptr;
  }
  return (length);
}
```

```
/*****************************************************************/
/*                                                               */
/*    a n y _ q u e u e _ e m p t y                              */
/*                                                               */
/*****************************************************************/
BOOLEAN   any_queue_empty(void)
{
 PROCESSOR          *nodeP;
 int                idx;

 idx         = 0;
 if (process_count/njobs > 0.60)
   while (idx < total_pe)
   {
    nodeP  = processor_table[idx].node_adr;
    if (feof(nodeP->job_file)
      && (nodeP->firstjb_ptr == NULL))
      return (TRUE);
    idx++;
   }
 return (FALSE);
}


/****************************************************************/
/*    i n t _ d i v i s i b l e                                 */
/*                                                              */
/****************************************************************/
BOOLEAN  int_divisible(double dividend, float divisor)
{
 if ((((int) (dividend / divisor)) * divisor)
    == dividend)
    return TRUE;
 else
    return FALSE;
}

/****************************************************************/
/*    G E T _ T I M E                                           */
/****************************************************************/
char *Get_Time(void)
{
 time_t clock;
 time (&clock);
 return (asctime(gmtime(&clock)));
}
```

```
/*****************************************************************/
/*   r a n d m c _ i d                                           */
/*****************************************************************/
int randmc_id(PROCESSOR   *nodeP)
{
 int             mc_id;

 do
 {
  mc_id = (int) (erand48(nodeP->xsubi) * total_pe);
 }while (mc_id  == nodeP->node_id);
 return (mc_id);
}

/*****************************************************************/
/*                                                               */
/*   E X I T _ D U M P                                           */
/*                                                               */
/*****************************************************************/
void exit_dump    (PROCESSOR *nodeP, USER_PROCESS  *procP)
{
#ifdef DEBUG_DUMP
  fprintf ( nodeP->trace_file, "\n\nProcess died at %.2f sec\n",
        nodeP->sys_real_time / ONE_SECOND);
  fprintf (nodeP->trace_file, "-------------------------------\n\n");

  fprintf (nodeP->trace_file, "Exec %.2f sec  Exist  %.2f sec\n",
        procP->exec_time / ONE_SECOND,
        procP->exist_time / ONE_SECOND);

  fprintf (nodeP->trace_file, "Response Ratio = %.2f sec\n\n\n",
        procP->exec_time / procP->exist_time);
#endif
}
```

```
/**************************************************************/
/*                                                            */
/*   P E R F O R M A N C E _ D U M P                          */
/*                                                            */
/**************************************************************/
void  performance_dump( PROCESSOR *nodeP)
{
 if (nodeP != NULL)
 {
  fprintf (nodeP->trace_file,
        "\nPerformance dump at  %.2f sec\n",
        nodeP->sys_real_time / ONE_SECOND);
  fprintf (nodeP->trace_file, "no. of procs %d\n\n",
        nodeP->load);
  if (nodeP->n_deaths != 0)
    fprintf (nodeP->trace_file, "Ave RT = %.2f sec\n",
          (nodeP->cumul_exist_time /ONE_SECOND) /
          nodeP->n_deaths);
  else
    fprintf (nodeP->trace_file, "Ave RT = 0.0 sec\n");

 } /* end of outermost IF */

} /* end of performance dump */


void Int_Trace(int code, ...)
{ int         i;
 PROCESSOR    *nodeP;

 fprintf (monitor_file,
       "\n --- SIMULATION DUMP PPC:%d --- @%s  \n",
       process_count, Get_Time());
 final_compute();
 Dump_RTime();
/*
 for (i = 0; i < total_pe ; i++)
 {
  nodeP      = processor_table[i].node_adr;
  NodeQ_Dump (nodeP);
  procQ_List (nodeP);
 }
*/
 fprintf(monitor_file, "\n\n");
 fflush(monitor_file);
 signal (SIGINT, Int_Trace);
}
```

```
void Int_Skip(int code, ...)
{  int   Sec;

  fprintf (monitor_file,
         "\n --- SKIPPED SIMULATION PPC:%d --- @%s \n",
         process_count, Get_Time());
  Sec = (int) (global_clock / ONE_SECOND);
  if (Sec > 0)
  {
   final_compute();
   SkipRun = TRUE;
  }
  fflush(monitor_file);
  signal (SIGUSR1, Int_Skip);
}
```

```
/* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   This is the main user-interface module for general menu-interaction,
   initialisation and initiation of simulation runs.

   However, this version differs from the previous( common.src) by
   abandoning the use of pregenerated, multiple jobfiles and their
   simulation. "Page Number"/buffer_no is used to simulate
   data generation through demand_paging.

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% */


#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <strings.h>
#include "model_params.h"
#include "model_types.h"
#include "globvar.h"

extern NODE_ENTRY     *processor_table;

/*****************************************************************/
/*     A collection of functions to process a dynamic               */
/*     network data structure.                                      */
/*****************************************************************/


FILE          *seedfile,
              *trace,
              *monitor_file,
              *configfile,
              *paramfile;


int            total_pe,
              mesh_len,
              positive_response,
              njobs,
              probe_limit,
              lba_index,
              skp_pid;

float          load_value,
              threshold;

char      *menu_1[] = {"1.   EXTEND  NETWORK\n",
                 "2.   REDUCE  NETWORK\n",
                 "3.   DISPLAY  NETWORK\n",
                 "4.   SIMULATE NETWORK\n",
                 "9.   EXIT PROGRAM\n\n\n"};

char      *menu_2[] = {"1.   DISCONNECT ROW\n",
```

```
                "2.   DELETE    ROW\n",
                "3.   DELETE    NODE\n",
                "4.   DELETE    ARC\n",
                "9.   RETURN TO MAIN\n\n\n"};

char     *menu_3[] = {"1.   GENERATE NEW ROW\n",
                "2.   MESH    ROW A & B\n",
                "3.   ADD     PATH\n",
                "4.   ADD     NODE\n",
                "9.   RETURN TO MAIN\n\n\n"};


/* ========================================================== */
/*   main - Test harness for network processing functions.           */
/* ========================================================== */
main (void)
{
/* declare all local variables.                    */

  int          menu_id,
             new_menu  = 1,
             option, batch_mode;

  BOOLEAN        exit_loop;

  void          run_simulation(void);
  int           menu_selection(int);

  /* main:  process code.                          */

  batch_mode    = BATCH_RUN;
  if (!batch_mode)
  {
   monitor_file  = stdout;
   do
   {/* select option and perform function                 */
    menu_id  = new_menu;
    option   = menu_selection(menu_id);
    switch (menu_id)
    {/* perform selected function                  */
      case 1: /*       */
           switch (option)
           {/* perform selected function                 */

             case 1: /*   */
                  new_menu  = 3;
                  break;

             case 2: /*   */
                  new_menu  = 2;
                  break;

             case 3: break;

             case 4: run_simulation();
                  break;

            default: /* exit from program                  */
                  break;
```

```
        }/* switch */
        break;

      case 2: /*      */
          switch (option)
          {/* perform selected function                       */

             case 1: /* push a given data item onto stack        */
                  break;
             case 2: /* pop a data item off the stack           */
                  break;
             case 3: /* display the contents of the stack       */
                  break;
             case 4: break;
            default: /* exit from program                      */
                  new_menu = 1;
                  break;
          }/* switch */
          break;

      case 3: /* display the contents of the stack         */
          switch (option)
          {/* perform selected function                       */

             case 1: /* push a given data item onto stack        */
                  break;
             case 2: /* pop a data item off the stack           */
                  break;
             case 3: /* display the contents of the stack       */
                  break;
             case 4: break;
            default: /* exit from program                      */
                 new_menu = 1;
                 break;
          }/* switch */
          break;

      default:/* exit from program                     */
          break;

    };/* switch */
    exit_loop = (menu_id == 1) && (option == 9);
   } while (!exit_loop);
  }
  else
   { char    trcName[15];
     skp_pid  = getpid();
     sprintf (trcName, "Pid%d.trc", skp_pid);
     monitor_file = fopen (trcName, "w");
     run_simulation();
   }
  fclose (monitor_file);
} /* main */
```

```
/* ============================================================ */
/*     menu_selection - function to display menu and return      */
/*               index of selected option                         */
/* ============================================================ */
int   menu_selection(int menu_id)
{ /* implementation */

  char      choice;
  void      Display_menu(char *[], int);

   switch (menu_id)
   {/* perform selected function                          */

     case 1: Display_menu(menu_1, 5);
           break;
     case 2: Display_menu(menu_2, 5);
           break;
     case 3: Display_menu(menu_3, 5);
           break;
     default: /*  */
           break;

    }/* switch */
  choice = getc(stdin) - 48;
  if ( (choice < 1) || (choice >9)  )
    return EOF;
  else return choice;
}/* menu_selection */


/* ============================================================ */
/*     Display_menu                                               */
/*                                                                */
/* ============================================================ */
void  Display_menu( char *menu[], int entries)
{
  int       index;

  for (index = 0; index < entries; index++)
     printf ("%s\n",menu[index]);

}
```

```
/* ================================================================ */
/*     run_simulation                                               */
/*                                                                  */
/* ================================================================ */
void run_simulation (void)
{
  extern void    create_config_file(int),
                  create_params_file(int),
                  simulate_network(void);

  extern void    Build_Network(),
                  write_intro_script();

  BOOLEAN    positive_response = FALSE;

  void       Initialise_Netmodel(NODE_ENTRY []),
             clear_workspace(NODE_ENTRY []),
             init_model_params(void);

  char        *Get_Time(void);

  int         run_count  = 1,
              prev_total = 0;

  fprintf (monitor_file,
        "\n *** SIMULATION COMMENCED ON %s \n\n", Get_Time());
  trace = fopen("main.trc", "a");
  create_config_file (BATCH_RUN);
  create_params_file (BATCH_RUN);
  configfile = fopen ("config.dat", "r");
  write_intro_script();
  while (!feof(configfile))
  {
   processor_table = (NODE_ENTRY *)
              calloc (total_pe, sizeof(NODE_ENTRY));
   fscanf (configfile, "%d", &mesh_len);
   if (prev_total == total_pe)
   {
    fscanf (configfile, "%d", &positive_response);
   }
   else
      prev_total = total_pe;
   Build_Network();
   seedfile  = fopen("seedfile.dat", "r");
   paramfile = fopen ("pramfile.dat", "r");
   fscanf (paramfile, "%f", &load_value);
   while (!feof(paramfile))
   {
    fprintf (monitor_file,
          "\nSIMULATION no.%d STARTED ON %s",
          run_count, Get_Time());
    Initialise_Netmodel(processor_table);
    init_model_params();
    signal (SIGINT, Int_Trace);
    signal (SIGUSR1, Int_Skip);
    simulate_network();
    fprintf (monitor_file,
          "-----------------------------------------------\n\n\014");
```

```
      fflush (monitor_file);
      fflush (trace);
      run_count++;
      fscanf (paramfile, "%f", &load_value);
    }
    fclose (paramfile);
    fclose (seedfile);
   clear_workspace(processor_table);
    write_intro_script();
  }
  fclose(configfile);
  fclose(trace);
}


/* ============================================================ */
/*     clear_workspace                                          */
/*                                                              */
/* ============================================================ */
void  clear_workspace (NODE_ENTRY processor_table[])
{
   int        idx;

   for (idx = 0; idx < total_pe; idx++)
   {
      cfree ((char *) processor_table[idx].node_adr->links);
      free ((char *) processor_table[idx].node_adr);
   }
   cfree ((char *) processor_table);
}
```

```
/* =============================================================== */
/*    Initialise_Netmodel                                          */
/*                                                                 */
/* =============================================================== */
void  Initialise_Netmodel(NODE_ENTRY processor_table[])
{ /* implementation */
  PROCESSOR   *nodeP;
  extern double drand48(void);
  int         idx1, idx2;

  for (idx1 = 0; idx1 < total_pe; idx1++)
  {
     nodeP = processor_table[idx1].node_adr;
    /* Construct data node */
    processor_table[idx1].await_sync = TRUE;
    nodeP->load          = N_SYS_PROCS;
    nodeP->pwait_set      = 0;
    nodeP->low_t_set      = 0;
    nodeP->high_t_set     = 0;
    nodeP->rplimit        = 0;
    nodeP->n_deaths       = 0;
    nodeP->n_local_procs   = N_SYS_PROCS;
    nodeP->n_virtual_procs  = 0;
    nodeP->n_tx           = 0;
    nodeP->n_rx           = 0;
    nodeP->n_migrates      = 0;
    nodeP->n_immigs       = 0;
    nodeP->cumul_exist_time = 0.0;
    nodeP->sys_real_time   = 0.0;
    nodeP->last_perf_dump_time  = 0.0;
    nodeP->next_elapsed_second  = 1000000.0;
    nodeP->n_active_local_procs = N_SYS_PROCS;
    nodeP->quanta_idx      = 0;
    nodeP->OSoverhead      = 0;
    for (idx2 = 0; idx2 < NQUANTA; idx2++)
    {
      nodeP->quanta[idx2].actual_load  = N_SYS_PROCS;
      nodeP->quanta[idx2].virtual_load = 0;
      nodeP->quanta[idx2].OSportion    = 0;
      nodeP->quanta[idx2].used        = 0;
    }
    for (idx2 = 0; idx2 < 3; idx2++)
    {
      do {
        nodeP->xsubi[idx2] = (int) (drand48() * total_pe);
      }while  (nodeP->xsubi[idx2] == nodeP->node_id);
    }
    for (idx2 = 0; idx2 < CSET_SIZE; idx2++)
    {
       nodeP->cSet[idx2].id = -1;
       nodeP->cSet[idx2].ld = 0;
    }
    nodeP->buffer_no   = 0;
    nodeP->curr_process= NULL;
    nodeP->firstjb_ptr = NULL;
    nodeP->lastjb_ptr  = NULL;
    nodeP->queue.tail  = NULL;
    nodeP->queue.head  = NULL;
    nodeP->timq.minP   = NULL;
```

```
    nodeP->timq.itype  = 0;
    nodeP->timq.mintime = 0;
   }
}


/* ============================================================ */
/*     init_model_params                                         */
/*                                                               */
/* ============================================================ */
void   init_model_params(void)
{
 extern void  job_creation(short int);
 char       *lba_text(void);
 int        idx;

   fscanf(paramfile, "%d", &njobs);
   fscanf (paramfile, "%d", &positive_response);
   fscanf(paramfile, "%d", &lba_index);
   if (positive_response)
   {
         if (system ("rm jobs*")) ;
         job_creation (0);
   }
   fprintf (monitor_file, "\n%15s%9d\t", "No. of Processors = ", total_pe);
   fprintf (monitor_file,  "%15s%1.2f",  "Load value       = ", load_value);
   fprintf (monitor_file,  "\n%15s%9d\t","Total No. of jobs = ", njobs);
   fprintf (monitor_file,  "%15s%s\n",   "Ld Balancing Alg. = ", lba_text());
   fscanf (paramfile, "%f", &threshold);
   for (idx = 0; idx < total_pe; idx++)
     (processor_table[idx].node_adr)->threshold = (int) threshold;

   if (lba_index > 0)
   {
    fscanf (paramfile, "%d", &probe_limit);
    fprintf (monitor_file, "\n%15s%9.2f\t",
                "Threshold Level   = ", threshold);
    switch (lba_index)
    {/* perform selected function  */
     case 2:  fprintf (monitor_file, "%15s%9d",
           "Probe Limit       = ", probe_limit);
           break;
     case 3:
     case 4:
     case 5:
     case 6:
     case 7:
     case 8:
     case 9:
     case 10:  fprintf (monitor_file,  "%20s%2d",
                "Distance         = ", probe_limit);
     default: break;
    }
   }
   fprintf (monitor_file, "\n%15s%1.2f%%\n","Convergence to < ",
                   100 * CONVERGE_FACTOR);
   fprintf (monitor_file, "\n");
 }
```

```
/* ============================================================== */
/*    l b a _ t e x t                                          */
/*                                                             */
/* ============================================================== */
char    *lba_text(void)
{

   switch (lba_index)
   {/* perform selected function                    */

    case 0: return ("No Load Balancing\0");

    case 1: return ("Random L/B Policy\0");

    case 2: return ("Threshold L/B Policy\0");

    case 3: return ("Reverse Broadcast\0");

    case 4: return ("Threshold Broadcast\0");

    case 5: return ("Threshold Nbor Policy\0");

    case 6: return ("Adaptive Th. Bcast\0");

    case 7: return ("Adaptive Rev. Bcast\0");

    case 8: return ("Global Avg Nbor Policy\0");

    case 9: return ("Global Avg Broadcast\0");

    case 10: return ("Global Avg Rcv Bcast\0");

    default: return ("L/B policy UNDEFINED\0");

   }/* switch */
}
```
